NEAREST NEIGHBOR SEARCH FOR CYRO-ELECTRON MICROSCOPY IMAGES

BY

YANG LIU

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2018

Urbana, Illinois

Adviser:

Assistant Professor Zhizhen Zhao

ABSTRACT

Cryo-electron microscopy (EM) technology enables researchers to determine the structure of a molecule at a much higher resolution than ever before. The single particle reconstruction (SPR) is one of the most widespread techniques used to reconstruct the 3D model of a molecule from its large set of cryo-EM projection images with unknown viewing angles. Classifying those projection images with similar views is regarded as one of the significant steps in the SPR problem. The goal in our research thesis is to find an effective way to identify and classify cryo-EM images solely based on their viewing angles.

In the first part of the thesis, we try four different k-nearest neighbors search (KNNS) algorithms on cryo-EM projection images classifying them based on the viewing directions. Brute force search (BFS) could be very time consuming when the dataset is large. Therefore, strategies like locality sensitive hashing (LSH) and some neural network methods are taken into consideration. The LSH method maps similar items into the same cell in a hash table, which effectively shortens the query time of searching for k-nearest neighbors, while the neural network methods are able to capture more expressive features, and then the classification is performed according to these learned features. The four KNNS algorithms include hyperplane LSH, cross-polytope LSH, unsupervised stochastic generative hashing (SGH) as well as unsupervised deep hashing (DH). Their performances are analyzed by the accuracy for finding 100 near neighbors for each query data, construction time and query time. However, all four algorithms fail to accurately classify projection images under similar views because of the orientation difference. Thus, further preprocessing techniques are required. One of the techniques we take is augmenting the image dataset by generating more planar rotation copies for each projection image in a way that allows more image candidates under the same view observable. It shows that with image augmentation, the

accuracies of all four methods increase. The LSH methods provide rather good performances both in computation time and the accuracy of searching for 100 nearest neighbors if the dataset is clean. However, for the noisy dataset, the traditional PCA is not working, and other preprocessing techniques should be applied to avoid color noise caused by the rotational image copies.

In the second part of the thesis, we attempt to devise a neural network model for learning rotation invariant features of the cryo-EM images as well as reconstructing them from the invariant features. After obtaining the desired rotation-invariant features, we can use the LSH method to do the classification since the trained features are distinguished solely depending on the viewing direction. In addition, this model possibly could be used in recovering the 3D molecule structure directly from 2D projection images, which will be explained in details in Section 3.3.7. However, since the image is often of large size, this 3D model reconstruction could be a highly complex computation problem. Currently, we begin with building up a simple version of this model and test with 1D signals. It shows that this model works well for small-length signals. However, when the signal length gets larger, it becomes harder to optimize the model and the result is less satisfying. Therefore, in the future we may need to adjust the architecture of the neural network model and tune the hyper-parameters to make the model adapt to a more general case.

To my parents and all I love, for their support.

ACKNOWLEDGMENTS

I would like to thank Prof. Zhizhen Zhao, my research advisor for leading me through the entire research process. Her insightful comments greatly improve this thesis.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 Cryo-EM Reconstruction Background

Cryo-EM is considered the most advanced technology in attaining molecule structure at a very high resolution, which can hardly be achieved by other traditional methods. The 2017 Nobel Prize in Chemistry was awarded to three researchers who spent years developing cryo-EM technology. The technique is a breakthrough in biochemistry. With the access of an atomic resolution for biomolecules, a better visualization of the molecule structures is available. This is critical in both the understanding of life's chemistry as well as the development of pharmacy.

The key of the cyro-EM technology is its ability to quickly and effectively trap many samples of a biomolecule in a thin layer of vitrified water, an amorphous solid that helps to retain the natural status and organization of the molecule. If the biomolecule is frozen too slowly, it will crystalize in a way that ruins its structure. Now with the aid of the cryo-EM technique, researchers are able to look at biomolecules in a near native state observing their movements as they function. After biomolecule samples are frozen, the microscope electron beam will be exploited to interact with them projecting multiple 2D images onto the detector. These generated projection images reveal the shape of the biomolecule, and are later used in determining the 3D structure of the biomolecule sample. The basic procedure for the 3D molecule reconstruction is demonstrated in Figure 1.1

The single particle reconstruction (SPR) [1] is one of the common strategies applied in the 3D molecule reconstruction problem, given a large set of 2D cryo-EM projection images. However, as the signal-to-noise ratio (SNR) of raw projection images is very high, it is almost impossible to directly use those images in the reconstruction procedure. In order to amplify the SNR,

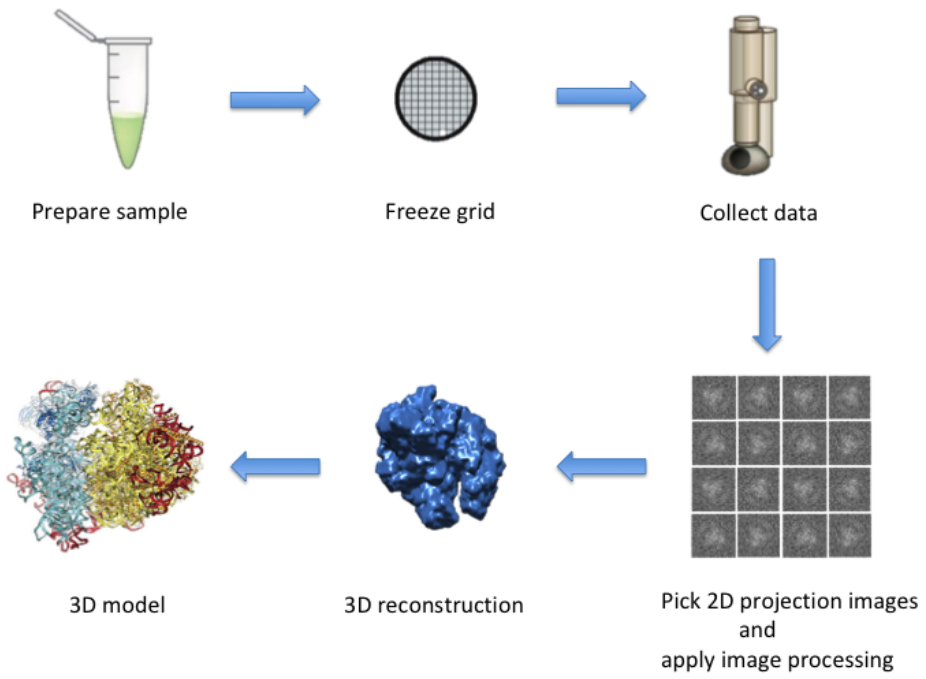


Figure 1.1: The process for 3D structure reconstruction of a human mitochondrial large ribosomal subunit with the use of cryo-EM technique. This figure is revised according to the figure from https://www.nature.com/articles/nmeth.3700.pdf?origin=ppub. The 3D model image is download from https://phys.org/news/2010-12-eukaryotic-ribosome-unveils.html

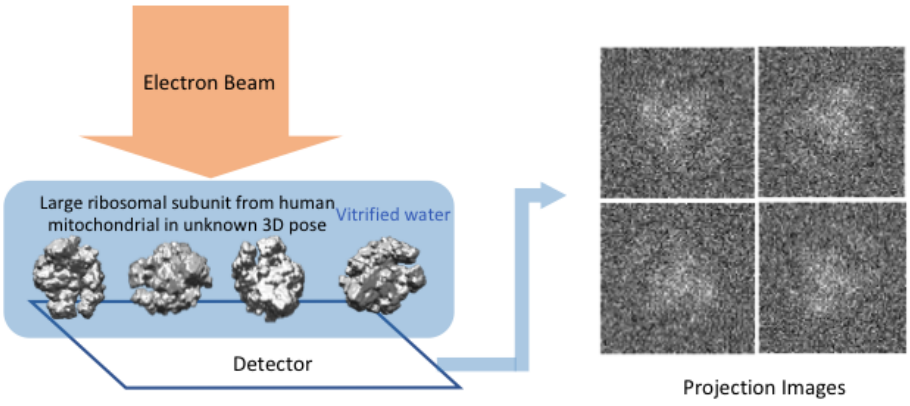


Figure 1.2: The process of generating cryo-EM projection images.

the class averaging method [2] is used in SPR, which generates sets of averaged de-noised images. The first step of this method requires classifying projection images with similar views. However, at the moment of imaging, the highly intense electron beam destroys the molecule structure. Therefore, a molecule can only be imaged once, and it becomes impractical to take projection images of the same molecule at different angles. But, same type of molecules share very similar structures, and we could instead assume they have the same structures. In this case, those 2D projection images gathered from different molecule samples of the same type could be used in SPR. Unfortunately, the collected cryo-EM projection images are randomly oriented and positioned because they could be in different poses when molecules are projected as shown in Figure 1.2. Further strategies need to be implemented for grouping images simply based on their viewing directions.

Our main task in this thesis is looking for a strategy that is able to effectively identify projection images with similar views regardless of angular orientations. This problem could be converted to a problem related to capturing the rotation-invariant features of the projection images and then performing the classification.

1.2 Previous Work

Searching data points based on their similarity is taken as a k-nearest neighbor search (KNNS) problem. The KNNS problem is a fundamental task that has been applied in numerous areas such as document retrieval, data clustering, machine learning and so forth. Mathematically, KNNS is defined as follows: given a set S of points in a space M, find the closest k points $p_1, p_2, ..., p_k$ in S to a query point $q \in M$. Various methods targeting the KNNS problem have been proposed. Their performance is evaluated by the time consumption of querying, the space complexity of the search data structures as well as how well the accuracy is achieved.

Generally speaking, there are two categories of methods for the KNNS problem - the exact method and the approximation method. The exact method has a naive linear searching time without any preprocessing steps [3]. It could become very time consuming if dataset size is very large or data dimension is very high. For such reasons, the approximate method is often

preferred in the real problem solving. In the approximate method, given a set S of points in a space M, the points returned whose metric distance from the query q is at most a times the metric distance from q to its real nearest point, where a is a small constant. It is shown that under general situations, the approximate nearest neighbor search methods can give a similar accuracy as the exact search methods but with a faster query time or with less memory space for maintaining the data structure.

Furthermore, there are mainly two ways to conduct the approximate nearest neighbor search. One is dividing the dataset into subsets based on the similarity and then performing the search, while the other is extracting the data features that preserve their structural similarity and then making comparisons among the obtained feature set. For the first way, a popular example is called locality sensitive hashing (LSH) [4, 5], an approach offering sub-linear query time and sub-quadratic space complexity. Hyperplane LSH [6] proposed by Charikar et al. gives a very good performance in practice; however, this method lacks sufficient theoretical supports [7]. We compare this method with cross-polytope LSH [7]. Cross-polytope LSH is proposed by Andoni et al., which tries to overcome the gap between the optimal performance in theory [8, 9] and the actual less satisfying performance provided in practice [10].

The other direction in solving the approximate nearest neighbor search problem is the neural network method. This method trains a network to optimize the mapping functions which are used to extract useful feature information from the dataset. In our experiments, methods include unsupervised stochastic generative hashing (SGH) [11] and unsupervised deep hashing (DH) [12]. SGH gets a handle on the relaxation problem and generates binary codes maximally compressing the dataset, while DH could capture the nonlinear manifold structure of data samples. We anticipated those neural network methods could help us to obtain the rotation-invariant features from the cyro-EM image dataset such that the projection images are classified by their views regardless of their randomly generated angular orientations and positions.

However, we found that all four methods are incapable to correctly identify the rotation-invariant features from the cryo-EM projection image dataset. For the LSHs, the measure metric used is either Euclidean similarity or cosine similarity, which only compares the element-wise difference between two data

samples and the rotation transformations of the image data is ignored. As for the neural network methods, output features are only translation invariant instead of rotation invariant. The accuracies for all four methods in finding the 100 near neighbors are very low. In order to boost the classification performance, we implement image augmentation in the data preprocessing step. By generating different rotation copies of each image sample, we provide more candidates to the searching methods. But this strategy is at the cost of memory space.

Looking for a more effective method, we are inspired by the TI-pooling method [13], which is a transformation-invariant pooling strategy using parallel siamese architectures. The ideal model we want to set up is capable of extracting the rotation-invariant features from the cryo-EM image dataset as well as reconstructing the 3D structure of a molecule based on its 2D projection images directly. Currently, only a toy model is proposed and tested with 1D signals. In the future, further developments will be dedicated to this simple model for more powerful functionalities.

1.3 Our Results

Two main experiments are done in this thesis. In the first one, we separately apply hyperplane LSH, cross-polytope LSH, unsupervised SGH and unsupervised DH separately on the original cryo-EM projection image dataset, and find that their accuracies for finding 100 near neighbors based on the viewing direction are very low. This result shows that these four methods fail to identify the rotational orientation similarity of images. After preprocessing the original image dataset with the image augmentation and using the augmented dataset instead, the accuracies of the four methods increase. In our experiments, we try different augmentation sizes of which augmented image samples are generated by rotating the projection image every m degrees, where m could be 5, 10, 40, and 90. Moreover, we find that the larger the augmented size is, the higher the accuracy could be. Among them, the LSH methods outperform the neural network methods on the querying and training time, while best accuracy varied depending on factors such as the augmentation size, de-noisy method used in preprocessing step and whether the dataset is noisy or not.

In the second experiment of our thesis, we attempt to build up a neural network to capture the rotation invariant feature of projection images, which the four methods failed to capture. We start with a simple model with an end-to-end architecture containing the encoding network, the average pooling and the decoding network. The encoding network consists of rotation transformation together with multiple fully connected layers. The width of the layer is larger than the length of the signal to avoid losing useful feature information of rotation copies. Furthermore, the decoding network takes the features obtained after the average pooling and processes them through another sets of fully connected layers in order to reconstruct the input signal back. The loss function compares the input and the reconstructed output from the decoding network minimized by back propagation. In the experiment, we first test with 1D signals of small lengths such as 10 or 41. It turns out that the intermediate features we learned from the network is rotational invariant. The network could reconstruct the input signal from the learned features providing an output in a form of the rotation copy of the original input. What is more, we test noisy input signals with different SNRs, and it works well when the signal length is small, e.g. 10. In addition, this model should be further developed and improved not only in classifying images with close views, but also in the 3D model reconstruction problem directly. However, the greatest challenge now is about the signal length. It seems that the loss rate of a signal of length 41 is stuck in some local minimums and cannot be improved further. For a more general situation, the architect of the neural network should be adjusted and other hyper-paremeter tuning methods should be tried in order to optimize the overall performance.

1.4 Thesis Organization

The rest of this thesis is organized into three chapters. In Chapter 2, we present four current advanced KNNS algorithms and then implement them in the cryo-EM image classification. Later, we compare and analyze their performances based on time complexity and 100 near neighbors accuracy. In Chapter 3, we propose a new idea about developing a rotation-invariant neural network and demonstrate this idea with a simple toy model. We then show the parameter adjustments in this neural network for obtaining better

performances in any one of clean, noisy and segmented input signals. In Chapter 4, we compare and analyze all the methods we use in the KNNS problem, evaluate the rotation-invariant neural network model and give the possible improvements we could try in the future.

CHAPTER 2

FAST K-NEAREST NEIGHBOR SEARCH FOR CRYO-EM IMAGES

2.1 Background Related Work

Nearest neighbor search is defined as that given a collection of data points S in space M and a similarity measure metric sim, for a query $q \in M$, find $p^* = argmin_{p \in S} sim(p, q)$. There are two main categories of methods for solving the nearest neighbor search problem: the exact search and the approximate search. However, due to the curse of dimensionality, the exact search method could be impractical such as k-d tree. Its complexity grows exponentially with the data dimension. Even after the image data is preprocessed and reduced to a lower dimensionality, for instance 256 in our experiment, it could still lead to 2^{256} partitions, which makes the query search extremely inefficient. In consequence, the approximate search method is more often applied in real problems. For this kind of method, hash functions are often used such that similar data will have high probability to be mapped into the same code, while two distinguishable data samples have low probability to be hashed into similar codes. Then, instead of comparing between complicated data samples, we compare their corresponding hashed codes.

Moreover, the existing hashing methods could be classified into two categories: data-independent [4, 14] and data-dependent [12, 15]. The first category uses random projections to hash data samples into binary code vectors, while the second category updates the parameters of hashing functions along with the training process and then converts data samples into codes by using the learned hash functions. A typical example for the first category is the locality sensitive hashing (LSH). This method is known for fast table construction and query search. The neural network method is a good example for the second category. It trains the hash functions to learn the intrinsic features from data, and later transform those data into representative codes

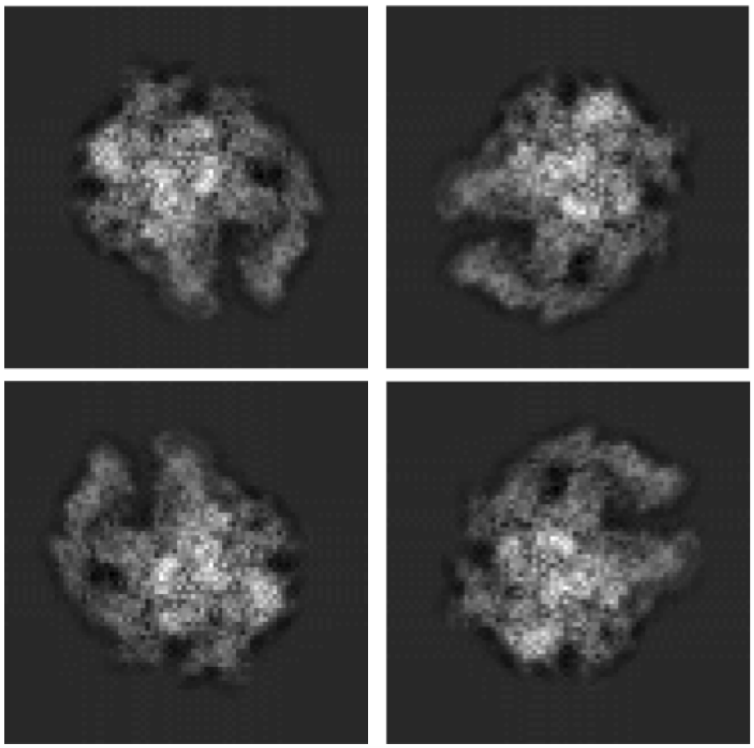


Figure 2.1: A sample image is rotated 90 degrees every time to generate a set of four image rotation copies.

in a more compact form.

In our research, we implement methods from both categories to classify cyro-EM projection images. At the end, we compare their time complexity as well as their accuracy of searching for 100 near neighbors.

Image Augmentation Image augmentation is a general technique that artificially expands the image dataset through various transformation methods, such as random rotations, translations, reflections and random cropping, etc. After implementing this technique, more image copies are provided to the KNNS method to boost the classification performance. In our experiment, since the problem is only related to in-planar orientation, the rotation transformation is applied while other transformations are trivial.

The orientation problem for the cryo-EM images is that image samples with similar views may not be categorized as near neighbors because of their orientation difference. In fact, if one of these two images is aligned by rotating a certain degree, it will look very similar to the other image. However, the four KNNS algorithms we try in this chapter could not treat image rota-

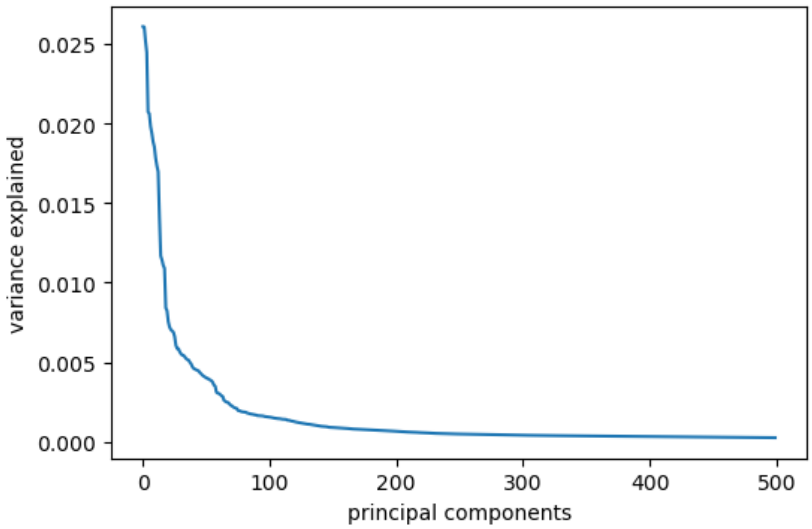


Figure 2.2: The ratio of variance explained by each principal component for the clean image dataset.

tional copies as images sharing similar viewing directions and fail to identify images with similar views as near neighbors. To solve this issue, we decide to augment the original dataset by adding more rotated copies for each projection image through image augmentation. In this way, the KNNS method is provided with more image candidates with different orientations under same viewing direction. In summary, the image augmentation method gives more information about the original projection images improving the classification accuracy, though at the cost of more training time and memory space for constructing the data structure.

In the experiment, we try the dataset augmented by 4 (each projection image is rotated by $360/4 = 90$ degrees every time), 9 (each projection image is rotated by $360/9 = 40$ degrees every time), 36 (each projection image is rotated by $360/36 = 10$ degrees every time), and 72 (each projection image is rotated by $360/72 = 5$ degrees every time). Figure 2.1 shows an example set of rotation copies with augmentation size 4. Those rotation copies which share the same viewing direction are considered as the same as their original image when we are calculating the accuracy for finding near neighbors. The functions we use to generate image rotational copies are from the ASPIRE toolbox.

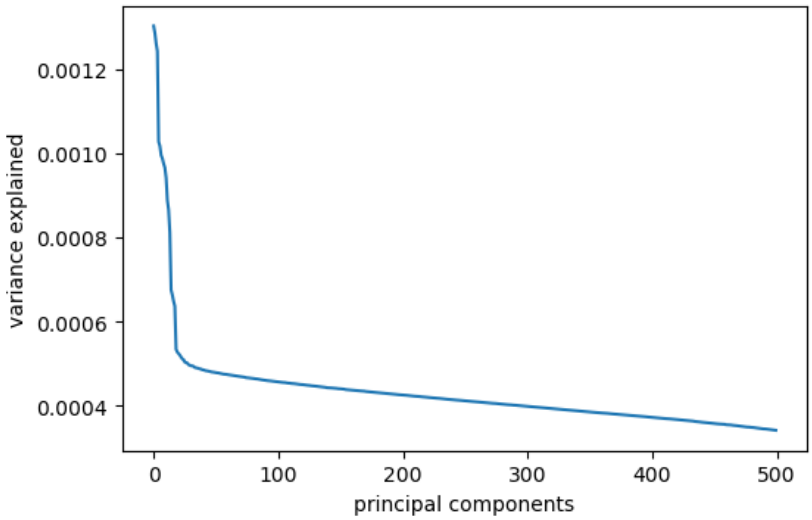


Figure 2.3: The ratio of variance explained by each principal component for the noisy image dataset.

Traditional Principle Component Analysis Principle Component Analysis (PCA) is often applied on one of the first steps [16] in the SPR problem. Because PCA could compress the image dataset onto a lower-dimensional space and at the same time extract principle information and discard less valuable information such as noise in projection images. Furthermore, it is also shown that in the cryo-EM problem, PCA outperforms single image based de-noising methods for example the Gaussian smoothing: 2D Gabor filter [17]. Therefore, PCA is implemented in our image preprocessing steps.

To be more specific, PCA simplifies the complexity in data of high dimension while still retains valuable features of the data. In the implementation procedure, a number of correlated data are transformed into a smaller number of uncorrelated data by projecting onto lower dimensions, which are defined as principle components (PCs). The first few PCs account for much variability in the dataset, while the remaining ones account for less variability. Generally speaking, most of the crucial information in an image is kept within the first few PCs, while the rest could simply be treated as trivial noise.

PCA uses Singular Value Decomposition (SVD) [18] to get PCs, and the number of PCs needed is determined by the corresponding singular values. When the singular values stabilize, the remaining values could be abandoned since those only represent trivial noise. Figure 2.2 shows the decreasing

singular values of the clean cryo-EM dataset, and they become stable after 200 PCs. So, we truncate the data dimension from 80×80 to 256. In addition, the noisy dataset shows a similar property as in Figure 2.3, and we also truncate its data dimension to 256.

Steerable Principle Component Analysis Because any planar rotation of the projection images could appear equally likely in the experiment by in-plane rotating either the specimen or the detector, we artificially expand the dataset by adding a certain number of planar rotation copies to each projection image, and then apply PCA to the augmented dataset. However, due to the correlated or say colored noise in each set of image rotation copies, spurious principal components occur when applying the traditional PCA, which leads to a lot of misclassifications. Therefore, we turn to the steerable PCA that is directly applied on the original dataset. The principal components of the steerable PCA consist of products of radial functions and angular Fourier modes [19, 20]. In this method, rotating a projection image is equivalent to adding phase shift to the expansion coefficients of the steerable PCA.

Fourier-Bessel steerable PCA (FBsPCA) [21] is one of the steerable PCA methods, which expands each image in the Fourier-Bessel basis. Since images are band limited in the Fourier domain, its expansion could be truncated into a finite series that preserves maximum amount of the feature information. Mathematically, an image could be expressed as $I(r,\theta) = \sum_{k,q} a_{k,q} v^{kq}(r,\theta)$ with coefficients $a_{k,q}$, where k is integer order for the Bessel function, q is the qth root of the equation $v^{kq}(r,\theta) = 0$. In theory, if an image I is rotated counterclockwise by α, the image is expressed as $I(r,\theta-\alpha) = \sum_{k,q} a_{k,q} v^{kq}(r,\theta-\alpha) = \sum_{k,q} a_{k,q} e^{-ik\alpha} v^{kq}(r,\theta)$. After applying a phase shift to the finite number of steerable PCA coefficients, we got the principal components for the rotation copy of the original image as well. Hence, rather than expanding the dataset by generating image rotation copies and then processing with the traditional PCA to compress and de-noise, we could directly use coefficients from FBsPCA and apply different phase shifts to generate different rotation copies for each projection image. This method is more time efficient and memory saving. The code for FBsPCA algorithm is also available in the ASPIRE toolbox.

Bispectrum-like Rotationally Invariant Image Representation [2] It will be more convenient if we can generate a rotationally invariant feature representation for each projection image, instead of preprocessing to create image copies like the traditional PCA or to create coefficient copies like FBsPCA. The bispectrum, a lossless shift-invariant representation is taken into consideration. It could be extended to produce rotation-invariant features for 2D images in the cryo-EM image classification [22]. This method is based on FBsPCA for which rotating the image amounts to phase shifting its relevant coefficients, and each image could be represented as $I(r,\theta) = \sum_{k,q} a_{k,q} v^{kq}(r,\theta)$ with coefficients $a_{k,q}$. Comparatively, the bispectrum of the steerable basis expansion coefficients is defined as $b_{k_1,k_2,q_1,q_2,q_3} = a_{k_1,q_1} a_{k_2,q_2} \overline{a_{k_1+k_2,q_3}}$, where k_1 and k_2 are angular indices and q_1, q_2, and q_3 are radial indices, and this bispectrum is proved to be rotational invariant. More details of proof are given by Zhao and Singer [2].

In our experiment, FBsPCA is performed at the beginning and its coefficients are used to generate the rotational invariant bispectrum-like features. The final feature dimension turns out to be 400. Though the dimension is slightly larger than the dimension obtained after applying the traditional PCA, its feature size is the same as the original dataset size rather than augmented size, which is quite decent.

2.1.1 Locality Sensitive Hashing

Locality sensitive hashing is a widely used method in the KNNS problem. The key of this algorithm is its hash functions which make pairs of similar data points more likely to be mapped into the same "bucket", while pairs of different points to different "buckets". A more formal definition is that given any two points $p, q \in \mathbb{R}^d$, by using a hash function $h \in F$, where F is a family of hash functions, p and q are treated as nearby when $h(p) = h(q)$ or far apart when $h(p) \neq h(q)$. The LSH family should satisfy the criterion that the probability of $h(p) = h(q)$ is at least P_1 if their distance is at most r_1, while the probability of $h(p) = h(q)$ is at most P_2 if the distance between them is at least $r_2 = c \times r_1$. The constant $c > 1$ represents the gap between "near" and "far", P_1 is the collision probability of nearby points, and P_2 is the collision probability of far apart points. Various kinds of hashing

functions are explored in the LSH method aiming for mapping the dataset more effectively.

After applying the hash function, the high-dimensional space $\mathbb{R}^d$ is partitioned into multiple "buckets" and data points are then assigned and stored in the corresponding "buckets". If the hash functions are good enough, finding near neighbors for the query point could be more efficient. Because, rather than searching in the entire dataset, we only need to search in the specific hash "bucket" that the query data is mapped into. Normally, the size of "bucket" is much smaller than the size of the whole dataset size leading to shorter querying time.

Hyperplane Locality Sensitive Hashing The hyperplane hash algorithm partitions the unit sphere $S^{d-1} \subset \mathbb{R}^d$ by randomly sampling a hyperplane cross the center of the sphere. This results in two halves of the sphere with equal size. Then, the normalized data point $v \in S^{d-1}$ is assigned to either side of the hyperplane. In the implementation, a random vector $r \in S^{d-1}$ is sampled, of which coordinates are i.i.d. Gaussian distributed. To hash point v, the dot product of v and r is computed. Whether the output hash bit should be -1 or +1 depends on the sign of the dot product. Mathematically, the procedure is defined as follows:

$$h_{\vec{r}}(\vec{v}) := \begin{cases} 1, & \text{if } \vec{r} \cdot \vec{v} \geqslant 0 \\ -1, & \text{if } \vec{r} \cdot \vec{v} < 0 \end{cases} \tag{2.1}$$

where $h_{\vec{r}}$ is one hash function generating one single output bit. After d' times of hashes, a d'-dimensional hash code will be generated for the data point v. However, choosing a random hyperplane is equivalent to generating a normally distributed random variable for each dimension, which means that even a single hash function representation requires a large number of random bits, and the procedure could be very computationally expensive. But with the help of the techniques [23, 24] proposed from Indyk and Engebretsen et al., for a number of n data vectors, $O(log^2 n)$ random bits are chosen for the hash functions such that the size of hyperplanes is upper bounded by $2^{O(log^2 n)}$.

In our experiment, each sample data is converted into a d'-dimensional hash codes consisted of either +1 or -1 for each entry. The KNNS for d-dimensional

data points is then transformed to searching for d'-dimensional hash code. In consequence, when comparing the similarity between data points in the base dataset and the query data point, we only calculate the Hamming distance of their hashed codes, which could be more computationally efficient.

Cross-Polytope Locality Sensitive Hashing The LSH algorithm could achieve a query time of $O(n^{\rho})$ and space complexity of $O(n^{1+\rho})$, where $\rho = \frac{1}{2c^2-1}$, and c is the constant representing the gap between "near" and "far" defined previously. However, it is not applicable in practice because its complicated hash functions that need a long time to be constructed, and sometimes even may be slower than a linear scan. Though the hyperplane LSH method works well in practice, it is said that the theoretical guarantees for it are not enough. Fortunately, the cross-polytope LSH method tries to solve the contradiction between the practical and theoretical issues for LSHs, and provides a good performance.

The cross-polytope hash algorithm hashes the data point v on the unit sphere $S^{d-1} \subset \mathbb{R}^d$ to the closest $\{\pm e_i\}_{1 \leqslant i \leqslant d}$ where e_i is the i-th standard basis vector in $\mathbb{R}^d$. The standard basis vector has exactly one nonzero coordinate that is either +1 or -1. During the procedure, a random matrix $A \in \mathbb{R}^{d \times d}$ with i.i.d. Gaussian entries is used to hash the point v. After computing $y = Av/||Av||$, we obtain the closest e_i to y from the set of standard basis vectors in $\mathbb{R}^d$.

However, it is extremely computationally expensive to use a truly random matrix. Suppose the data dimension is d, to multiply a random Gaussian matrix with this vector, $O(d^2)$ time is required. This is infeasible when d is very large. To solve this issue, the pseudo-random projection is taken into consideration. Rather than directly multiplying the input vector v with a random Gaussian matrix, we process with three sets of HD as $y = HD_3HD_2HD_1$, where H is the fast Hadamard transform [25], and D_1,D_2,D_3 are a diagonal matrixes with only ± 1 elements. By using the Fast Hadamard Transform, the evaluation time of this orthogonal transformation is brought down to $O(d \log d)$ and its space complexity is restricted to $O(d)$. It turns out that the multiplications of three sets of HDs is equivalent to directly applying a true random rotation matrix if d goes to infinity. However, it is also said that the application of less than three sets of HD is not sufficient.

The other important strategy cross-polytope LSH executed to boost the

performance is the multi-probe scheme [26]. In the standard LSH data structure, only one "bucket" in each hash table will be examined when searching for the query point. But, in the multi-probe LSH data structure, candidates from multiple cells in each hash table will be taken into account. Under such a situation, the query q closest to the point $p \in S^{d-1}$ but fails to collide with p under a certain hash function is still likely to be hashed to a value that is approximate. Even though q fails to be hashed into the correct "bucket", it is still possible to find q in the nearby "buckets" by utilizing the multi-probe scheme. Probing multiple hash locations in the same table at the same time quickens the query speed and provides a relatively high accuracy for finding near neighbors.

2.1.2 Neural Network Methods

Unsupervised Stochastic Generative Hashing The neural network method is a learning-based binary hashing method that convert inputs of high dimension into simple binary output while still largely maintains the crucial feature information. However, inferior performance may occur due to the relaxation problem caused by binary outputs for the hash functions. The stochastic generative hashing (SGH) method is brought up to handle this problem.

Most of the hashing methods model the forward process of creating binary codes from the input. On the contrary, the procedure of SGH is reverse. It generates inputs from output binary codes such that the generated hash codes could maximally preserve the local neighborhood structure in the original data space. What is more, this hashing method learns hash functions by the Minimum Description Length (MDL) principle that generates the learned hash codes which could maximally compress the dataset. By implementing the distributional stochastic gradient descent (SGD), the method avoids the difficulty caused by discrete output constraints, and at the same time optimize parameters of hash functions as well as the generative model.

Suppose this algorithm is trained with input samples $\{x_i\}_{i=1}^{N}$ of size N. First of all, the parameter set $\Theta_0 = \{W, U, \beta, \rho\}$ is initialized randomly, where W belongs to the encoding function $q(h|x)$, and U, β, ρ are parameters defined for the generative model $p(x, h)$. Secondly, the number of training iterations

is determined. For each iteration, the following procedure will be executed:

1. Sample batch of x_i uniformly from the training dataset $\{x_i\}_{i=1}^N$.

2. Sample auxiliary variables ϵ, ζ of the distributional SGD according to Gaussian distribution.

3. Compute the proposed distributional SGD which is

$$\hat{\nabla}_\Theta \widetilde{H}(\Theta_{i-1}; x_i) = [D_W \widetilde{H}(\Theta_{i-1}; x_i), \hat{\nabla}_{U,\beta,\rho} \widetilde{H}(\Theta_{i-1}; x_i)] \tag{2.2}$$

   where $\widetilde{H}(\Theta) = \sum_x \mathbb{E}_{\epsilon,\zeta}[l(\widetilde{h}, x)]$ and $l(\widetilde{h}, x) := log\ p(x, \widetilde{h}(\sigma(W^T x), \epsilon, \zeta)) - log\ q(\widetilde{h}(\sigma(W^T x), \epsilon, \zeta)|x)$. The $\widetilde{h}$ is a doubly stochastic neuron that gives a meaningful distributional derivative to $D_W \widetilde{H}(\Theta_{i-1}; x_i)$, of which the definition is shown as follows:

$$\widetilde{h} := \begin{cases} 1, & \text{if } z > \epsilon \\ \mathbf{1}_{z>\zeta}, & \text{if } z = \epsilon. \\ 0, & \text{if } z < \epsilon \end{cases} \tag{2.3}$$

4. Update the parameter set by $\Theta_i = \Theta_{i-1} + \gamma \hat{\nabla}_\Theta \widetilde{H}(\Theta_{i-1}; x_i)$, where γ is a predefined learning rate.

This model is end-to-end trainable of which architecture is similar to the architecture of the binary auto-encoder (BA) [27]. But the encoding procedure of BA is deterministic that may cause bits waste. Moreover, the optimization of BA is difficult due to the binary constraints, while this proposed method could overcome such difficulty thanks to the implementation of the stochasticity.

After the training procedure is completed, output binary codes for the input dataset and the query dataset could be simply compared by their Hamming distance. Near neighbors are identified from the pairs that return the minimum distances.

Unsupervised Deep Hashing Most current binary code learning methods use linear projection to map data samples into binary vectors such as LSHs which use random projection matrices to hash data into simple codes.

Comparatively, the unsupervised deep hashing method uses multiple hierarchical non-linear transformations that is able to capture the nonlinear relationship among the dataset.

Unsupervised deep hashing is a deep neural network that learns and optimizes parameters by three criterions including (1) minimizing the loss between the input samples and the learned binary vectors; (2) distributing each bit evenly for the generated binary codes; and (3) trying to assign each output bit independently. The network parameters are updated by the back-propagation based on the optimization objective function shown as

$$\begin{aligned} min_{W,b}L &= L_1 - \lambda_1 L_2 + \lambda_2 L_3 + \lambda_3 L_4 \\ &= \frac{1}{2}||C - H^M||_F^2 + \frac{1}{2N}tr((H^M H^M)^T) \\ &+ \frac{1}{2}\sum_{k=1}^{n}||W^m(W^m)^T - I||_F^2 + \frac{1}{2}(||W^m||_F^2 + ||b^m||_2^2) \end{aligned} \tag{2.4}$$

C is the matrix representation of the output binary vectors and H^m is the output from the mth layer of the network, where m is ranging from 0 to M. W^m is the weight of the mth network layer and b^m is the bias of the mth network layer. L_1 aims to minimize the quantization loss between output vectors and the original real valued vectors, L_2 tries to balance bits by maximizing the variance of learned binary vectors. L_3 helps to maximize the independence of each transform. L_4 are regularizers to adjust the parameters scales.

Taking the stochastic gradient descent, parameters $\{W_m, b_m\}_{m=1}^M$ are updated as below in each iteration until convergence.

$$W^m = W^m - \eta\frac{\partial L}{\partial W^m} \tag{2.5}$$

$$b^m = b^m - \eta\frac{\partial L}{\partial b^m} \tag{2.6}$$

After getting the parameters $\{W_m, b_m\}_{m=1}^M$, we could hash our input dataset and query dataset into binary codes, calculate their Hamming distance, choose the candidates that return minimum distances and claim as near neighbors.

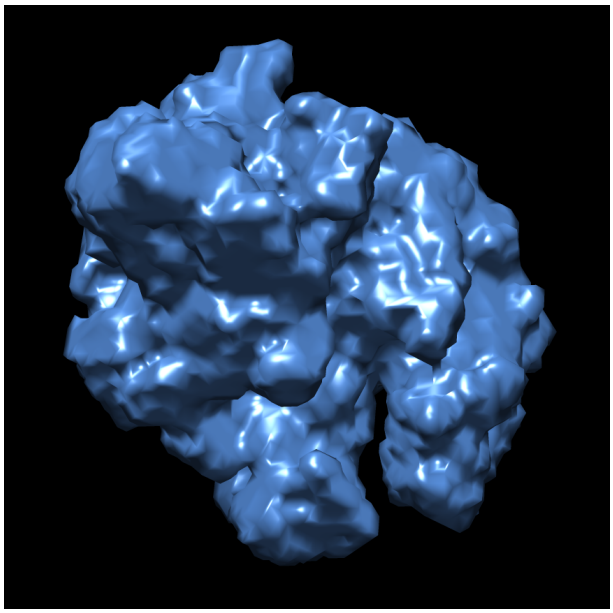


Figure 2.4: 3D structural model of a large ribosomal subunit from human mitochondria.

2.2 Experiment and Result

2.2.1 Image Types - Clean and Noisy

The molecule used in this thesis is a large ribosomal subunit from human mitochondria of which structure is displayed in EMD-2762 as Figure 2.4. In order to simulate the procedure of reconstructing a 3D molecule model from 2D projection images, firstly we need to generate 10000 80×80 cryo-EM projection images. Since the quaternion vector can represent the viewing direction and orientation of an object, 10000 quaternion vectors are randomly created and applied for projecting the molecule onto a set of 2D images by calling functions from the ASPIRE toolbox. Finally, 10000 centered clean projection images are generated, some of which share the similar viewing direction but have quite different orientations as the example shown in Figure 2.1.

In addition, it is noticed that the real cryo-EM images are very noisy with low SNR. Therefore, we also need to test KNNS algorithms on the noisy dataset to see how well they could perform. In the experiment, the SNR is set to 0.1. For comparison, the clean and noisy image data are shown in

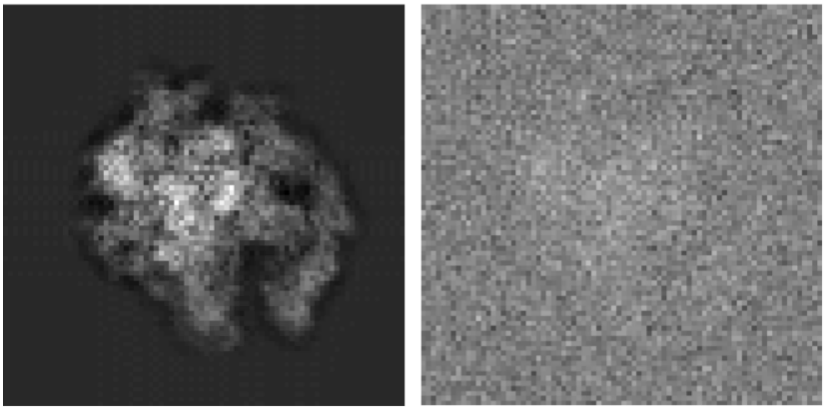


Figure 2.5: A clean cryo-EM image (left) and a noisy cryo-EM image with SNR = 0.1 (right).

Figure 2.5, and we could see that the image on the right-hand side is heavily contaminated by noise.

2.2.2 Dataset Description

A total of 10000 2D projection images are generated from the 3D structure model with respect to 10000 quaternion vectors. The first 9900 images are used for training procedure, and the remaining 100 images are used for testing procedure as query dataset. Furthermore, to avoid the misclassification caused by in-plane orientation, the training dataset is augmented by 4, 9, 32 and 72 times of the original size. These training datasets are later tested in the experiment. They are all preprocessed by the traditional PCA cutting down the data dimension from $80 \times 80 = 6400$ to 256.

In the following experiment, the same procedure is also done for testing the noisy dataset. However, it shows that the traditional PCA does not work well in the noise reduction for the noisy image dataset. As a result, we have to turn for other methods: FBsPCA and the bispectrum. The feature vector obtained after performing FBsPCA contains a set of complex numbers. Since all algorithms we tired so far only take in real numbers, we concatenate the values of real and imaginary part, which gives a twice length feature vector with all real numbers. Furthermore, we can just create rotational image feature representatives by phase shifting the original image feature gained after FBsPCA. This avoids augmenting the original image dataset and then apply the traditional PCA to the enlarged dataset, which saves memory space

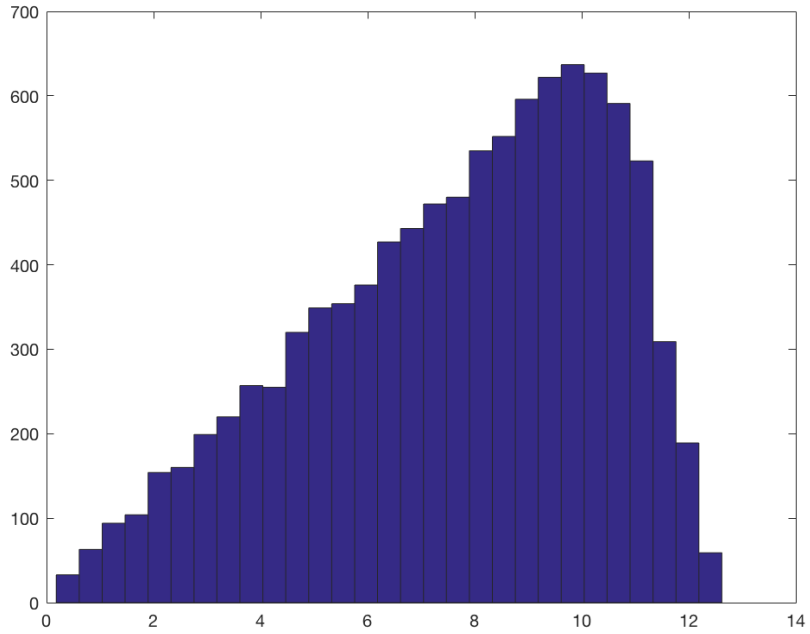


Figure 2.6: Histogram of 100 near neighbors for the whole query dataset.

as well as shortens the operating time. As for the bispectrum, it uses features obtained from FBsPCA and generates rotation-invariant features. In this way, the augmentation is no longer needed. Though the feature dimension is 400, it is effectively compress the data size of which the dimension is 6400.

2.2.3 Ground Truth Table

The quaternion vectors are used to create the ground truth table for 100 near neighbors for each query. The *q_to_d* function in the ASPIRE package is used to calculate the viewing angle difference between the query quaternion vector and one of quaternion vectors from the training dataset. Later, 9900 quaternion vectors are sorted according to the viewing angle difference in ascending order, and the first 100 are picked as the 100 near neighbors for the specified query quaternion vector. The same procedure is then applied to all other query quaternion vectors.

The histogram of viewing direction differences for 100 near neighbors of all queries is shown in Figure 2.6. We could see that the range of viewing angles is from 0 to 14 degrees and most of the angles are concentrated around 10 degrees.

Table 2.1: Hyperplane LSH accuracy vs. L

L (number of hash tables)	5	10	15	20	50
Construction time(s)	5.490	10.206	13.392	17.859	46.877
100 NNs accuracy	0.9158	0.9162	0.9162	0.9162	0.9162

Table 2.2: Hyperplane LSH accuracy vs. K and T

K	1	2	3
Optimal T	100	2500	10000
100 NNs accuracy	0.9157	0.9157	0.91

2.2.4 Training Scheme

In this section, algorithms including hyperplane LSH, cross-polytope LSH, unsupervised SGH, and unsupervised DH are tested with both the clean and noisy image datasets of different augmented sizes. In the following subsections, the training scheme for each method will be explained in detail.

LSHs Training Scheme The training process of LSH is constructing hash tables and storing data points in their corresponding "buckets" in those hash tables based on the similarity of data points. To be more specific, L different hash tables are generated and for each hash table K different hash functions are used. Suppose there are N training data points, $N/2^K$ points on average will be put into each "bucket". While searching near neighbors for a query point, we simply hash the query point into those hash tables and choose candidate points that share the same "bucket". In the experiment, code of hyperplane LSH and cross-polytope LSH is from the FALCONN package. Several parameters need to be tuned based on the properties of the dataset before the table construction. We change only one variable at a time while we fix others to examine how this specific parameter will affect the performance of the algorithm. The whole training procedure will be explain based on the clean dataset with augmentation size 72 in the following paragraphs.

For hyperplane LSH, three parameters are required to be adjusted: L-

Table 2.3: Hyperplane LSH parameters

Parameter	L	K	T
Number	10	2	2500

Table 2.4: Cross-polytope LSH accuracy vs. L

L (number of hash tables)	5	10	15	20	50
Construction time(s)	3.683	6.548	9.845	12.222	31.699
100 NNs accuracy	0.916	0.9158	0.9158	0.9158	0.9158

Table 2.5: Cross-polytope LSH accuracy vs. $number\_of\_hash\_bits$

$number\_of\_hash\_bits$	5	10	12	16	20
Optimal T	100	500	2500	2500	7500
100 NNs accuracy	0.9158	0.9158	0.9158	0.9162	0.9161

the number of hash tables, K-the number of hash functions per hash table, and T-the number of probes. L is the first parameter needed to be set up considering the available memory space in the computer. The larger the L value is, the more memory space is required and the longer time it takes to construct hash tables. The construction time and the corresponding accuracy for each L is shown in Table 2.1. From this table, we can see that when L achieves a certain number, increasing L will no longer further improve the accuracy result, and on the contrary, the table construction time will continue to rise. Thus, L is set to 10.

After deciding the L value, we need to choose K and T. These two parameters are required to be changed in pair, since the larger K is, the more T is desired for achieving a better performance. However, K could not be very large otherwise memory error will occur. In the experiment, we find the optimal T for each tested K. Since trying different T does not require reconstructing hash tables, the procedure is not time consuming. We use a binary search method to obtain the optimal T as mentioned in the FALCONN package. The result of optimal T for each different K is shown in Table 2.2. When K equals to 2 and T equals to 2500, the best performance for finding 100NNs is achieved. In the end, we find the set of parameters in Table 2.3 for hyperplane LSH could give a rather good accuracy result.

The similar training procedure is also done for cross-polytope LSH. However, the difference is that rather than simply adjusting K, another pa-

Table 2.6: Cross-polytope LSH parameters

Parameter	L	$number\_of\_hash\_bits$	T
Number	10	16	2500

Table 2.7: SGH parameters

Parameter	bit number	batch	initial learning rate	iteration number
Number	64	500	1e-2	5000

rameter named *last_cp_dimension* is also required to be changed at the same time. These two parameters could be determined by a function called *compute_number_of_hash_functions* of which input is *number_of_hash_bits*. It is suggested to set this input parameter the same order of magnitude as the number of training data points. Table 2.4 shows the construction time and its corresponding accuracy along with the change of L, of which performance is similar to the one got from hyperplane LSH. Too large L could not further improve the accuracy result but rather increase the construction time. In addition, Table 2.5 shows the optimal T under different *number_of_hash_bits*. It is found that when *number_of_hash_bits* is set to 16 and T to 2500 as in Table 2.6, the best accuracy performance for our dataset is attained.

SGH Training Scheme Comparing to LSHs, the training time of the SGH is much longer, since hash functions are learned along with the training procedure. The hash functions includes weights and bias of the network, which will be obtained after the training. Image data is then converted into binary code by applying the trained hash function.

To start with, several parameters require adjustment such as the batch size, the learning rate, the maximum of iteration number, the number of output hash bits and others helping to define the loss function. Normally, larger batch size could result in faster convergence, but at the cost of the memory space. It turns out that 500 batch size is good enough for our training procedure. As for the learning rate, too large a value could result in passing the minimum point for the loss function, while too small a value could lead to a very long training time. At the end, we find that setting the learning rate to 1e-2 could give us a good loss rate. What is more, 5000 training iterations is considered long enough for the loss rate to converge in our experiment. After trying several different combinations of parameters, we find that the set of parameters in Table 2.7 could provide a relatively high accuracy for finding 100 near neighbors.

Unsupervised DH Training Scheme Like SGH, the hash function of DH is also made up of weights and bias of the neural network, and are learned during the training procedure, which takes a longer time compared to LSHs. In the experiment, the only parameter we need to change is the number of output feature bits. It is chosen from 16, 32, 48, and 64, and shows that the more binary bits used for representing the feature information, the more accurate the results. This is because more representative bits could help to keep more useful information about the original data. Yet, too many bits may deteriorate the accuracy because some trivial information like noise is also captured in the redundant bits. Currently, 64-bit is enough for our dataset.

Table 2.8: Construction/Training time(s)

Method	x1	x4	x9	x36	x72
HP LSH	0.140	0.560	1.220	4.881	10.3312
CP LSH	0.09339	0.4224	1.0002	4.4807	9.7723
SGH	55.5592	59.6032	66.0665	74.0927	72.4245
DH	5.3942	8.2541	11.7402	38.9645	112.7794

Table 2.9: Query time(s)

Method	x1	x4	x9	x36	x72
BFS	0.01229	0.04048	0.17960	0.8435	1.4584
HP LSH	0.004548	0.005343	0.006244	0.008872	0.01598
CP LSH	0.002905	0.005061	0.006337	0.007963	0.005704
SGH	0.0086	0.0304	0.0655	0.2907	0.5859
DH	0.0122	0.0173	0.0305	0.1029	0.1957

Table 2.10: 100 near neighbors accuracy (%) for the clean image dataset

Method	x1	x4	x9	x36	x72
BFS	9.25	27.87	49.40	89.53	91.58
HP LSH	9.25	27.87	49.40	89.53	91.57
CP LSH	9.24	27.75	49.37	89.48	91.59
SGH	8.74	23.55	40.46	62.03	68.97
DH	8.74	25.17	43.26	74.10	76.59

2.2.5 Analysis

From Figure 2.7, we could see that with the original clean dataset, a large portion of near neighbors obtained are actually far away from the query data for all four methods due to the orientation problem. Therefore, these four methods are unable to correctly identify image of different orientations with similar views. However, when we increase the dataset by adding rotation copies for each projection image, the performances of four methods are all improved. The rotation copies are considered the same as the original image and only one is used as one of the 100 near neighbors in the accuracy calculation.

The construction time, query time and 100 near neighbors accuracy are all taken into account while we analyze the performance of these four algorithms. As shown in Table 2.8 and Table 2.9, cross-polytope LSH outperforms other three methods providing the least construction time and query time, which is very time efficient. (Note: the query time here is the average time that each query takes to get its 100 near neighbors in the training dataset.)

What is more, cross-polytope LSH gives the second-best performance only inferior to hyperplane LSH regarding the 100 near neighbors accuracy as we see in Table 2.10. We also notice that the accuracy of cross-polytope LSH rises faster than that of hyperplane LSH when the dataset size is increasing. When the dataset size increases to 72 times of the original size, the performance of cross-polytope LSH grows better than that of hyperplane LSH. In conclusion, cross-polytope LSH implemented on a dataset with augmentation size 72 is superior among all other tests we ran.

2.2.6 Problem with Noisy Dataset

Cryo-EM images have excessive white noise and clean images are actually not accessible. The SNR of cryo-EM images is very low, of which value is usually less than 0.1. After generating noisy dataset with 0.1 SNR, we tested it on all four KNNS algorithms to see how well they could perform in reality.

In this section, the way for calculating the accuracy is redefined. Instead of calculating the overlap candidates between the 100 near neighbors got from the specific KNNS method and those from the ground truth table, we check whether the near neighbors found for each query satisfy the criterion:

$\cos(q, p) \geqslant 0.9$, where q is the query point and p is one of the candidates from the training dataset. The reason we change the definition is that since the real image dataset is contaminated with high level of noise, small deviations for the viewing angle are acceptable. The new accuracy definition is less strict than the previous one, and the corresponding histogram result of query dataset is shown in Figure 2.8.

From Table 2.11, we see that even though the dataset is augmented 72 times, the accuracies obtained are all below 75%. This is mainly because the noise level is very high and the traditional PCA could not effectively deduce the correlated noise among the augmented image rotation copies. In this case, two other de-noising methods including FBsPCA and the bispectrum are tried in order to address this issue.

After preprocessing the original dataset with FBsPCA, we add different phase shifts to the output coefficients for each image data to get its corresponding rotation feature copies. The coefficients obtained for each image data are 40 complex numbers. Since a real value dataset is required for the four algorithms, the values from the real part and the imaginary part are concatenated into a new vector of length 80 for later classification. Since this method is implemented on the original dataset, it is fast in computation compared to the traditional PCA. From Table 2.12, we see that the performance of FBsPCA is better than that of the traditional PCA for the four methods. Moreover, when augmentation size is small, the accuracy after performing FBsPCA is much superior than the accuracy after performing the traditional PCA.

The bispectrum is another method we tried to preprocess the image dataset. After getting the coefficient outputs from FBsPCA, the $Bispec\_2Drot\_large$ function in the ASPIRE toolbox is called to compute the rotationally invariant bispectral-like features for the each projection image. For this method, augmentation is no longer necessary since the feature is rotation-invariant. The result is shown in Table 2.12 at the last column. We can see that it provides even better performances than ones got from FBsPCA with augmentation size 72. In general, the results obtained from the bispectrum method are more satisfying than those got from the traditional PCA and the FBsPCA even with augmentation size 72.

Table 2.11: $\cos(p, q) \geqslant 0.9$ accuracy (%) for the noisy dataset preprocessed by the traditional PCA

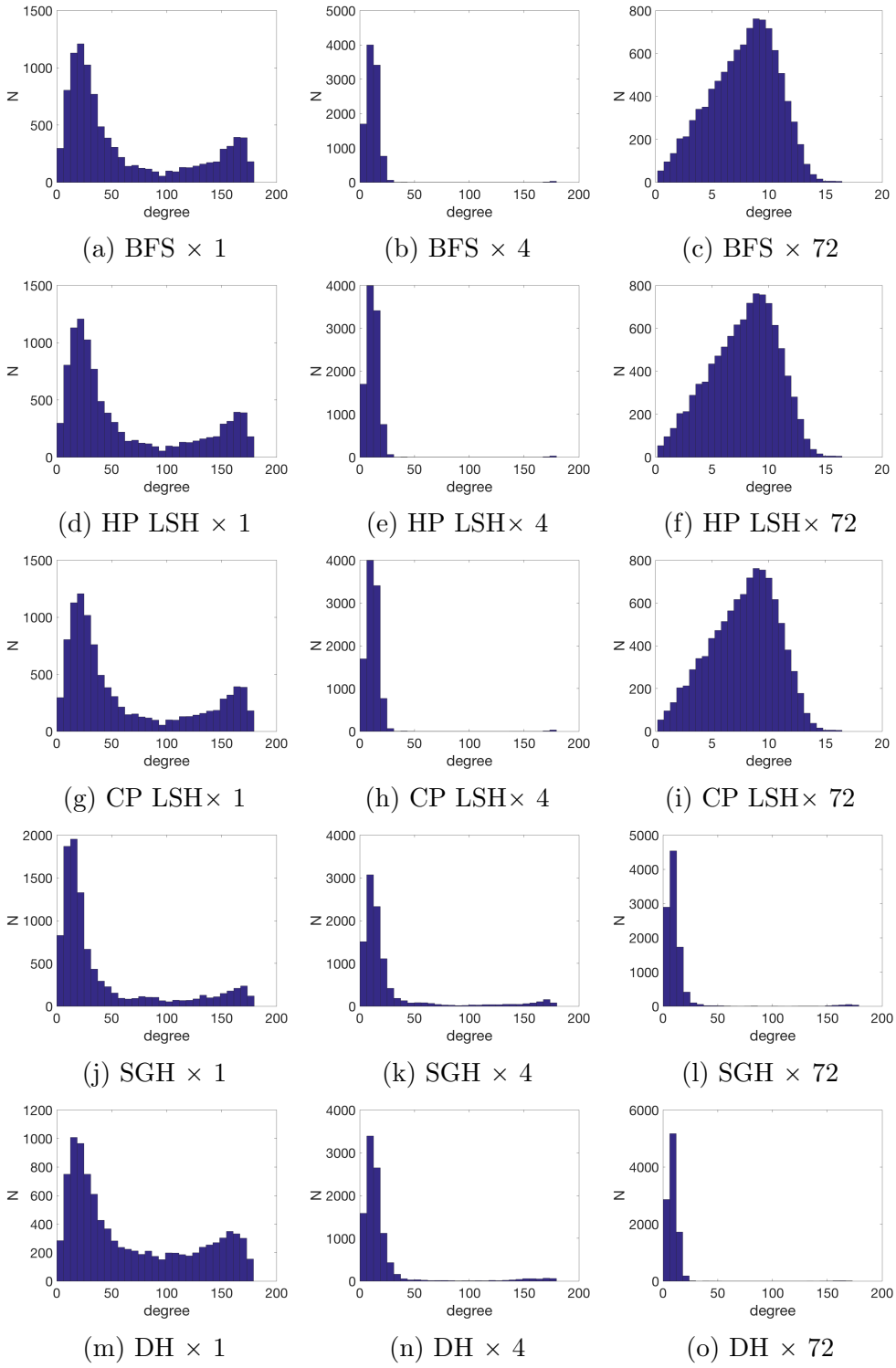
Method	x1	x4	x9	x36	x72
BFS	12.60	28.35	44.64	67.65	73.61
HP LSH	12.37	26.30	40.87	59.66	65.83
CP LSH	10.31	24.22	35.58	53.66	59.75
SGH	5.16	34.16	43.07	57.31	54.41
DH	13.89	27.64	41.15	57.93	62.35

Table 2.12: $\cos(p, q) \geqslant 0.9$ accuracy (%) for the noisy dataset preprocessed by the FBsPCA or the bispectrum method

Method	x1	x4	x9	x36	x72	bispectrum
BFS	31.83	51.33	61.98	68.43	69.58	70.62
HP LSH	31.83	51.35	61.98	68.50	69.58	70.52
CP LSH	31.92	51.38	62.01	68.41	69.54	69.68
SGH	26.38	44.87	55.52	69.04	71.40	64.26
DH	28.55	44.25	55.37	68.22	66.77	70.85

2.3 Summary

In this chapter, we find that for the clean dataset, cross-polytope LSH outperforms the other three methods in the construction time and the query time, while it still achieves a rather good result for the 100 near neighbor accuracy. However, in real application, the projection images is always highly contaminated with noise and this should be taken into account. For the dataset with 0.1 SNR, we observe that the result attained from preprocessing with the traditional PCA is not good. After comparing with the FBsPCA method and the bispectrum method, we find that the later method gives a comparatively good performance for all the algorithms, which accuracies could all achieve around 70%. The bispectrum-like feature gained from the bispectrum method is rotational invariant as well as denoising effectively. Yet, those accuracies are all below 75% and not considered as satisfying while are still needed to be improved in future work.



(a) BFS × 1 (b) BFS × 4 (c) BFS × 72

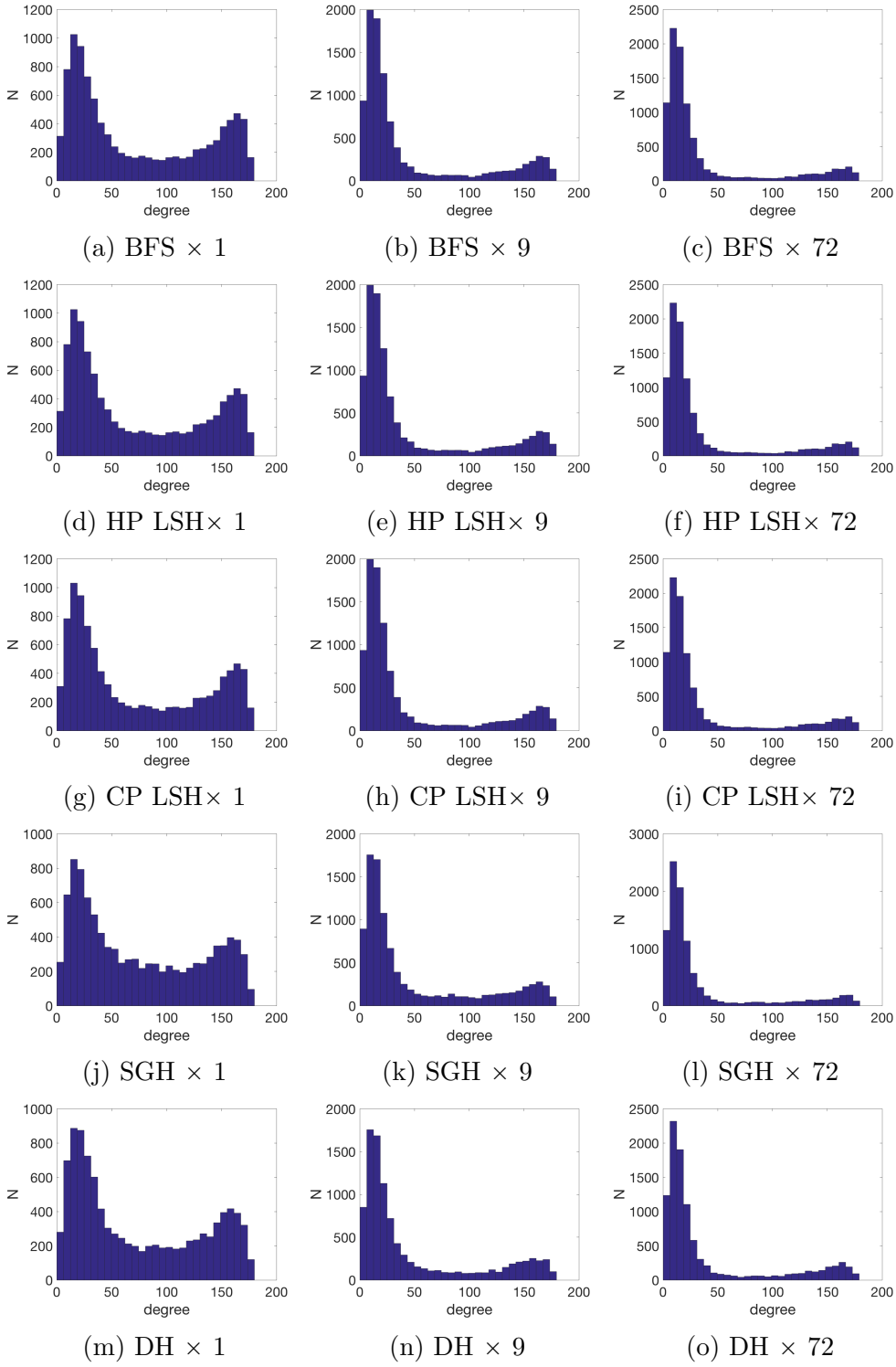
(d) HP LSH × 1 (e) HP LSH× 4 (f) HP LSH× 72

(g) CP LSH× 1 (h) CP LSH× 4 (i) CP LSH× 72

(j) SGH × 1 (k) SGH × 4 (l) SGH × 72

(m) DH × 1 (n) DH × 4 (o) DH × 72

Figure 2.7: Histogram of 100 near neighbors for BFS, hyperplane LSH, cross-polytope LSH, SGH and DH with clean training dataset of augmented data size 1, 4, and 72; the range of x-axis for (a),(b),(d),(e),(g),(h),(j),(k), and (l) is [0,180] degree, and the range of x-axis for (c),(f), and (i) is [0,18] degree.



(a) BFS × 1 (b) BFS × 9 (c) BFS × 72

(d) HP LSH× 1 (e) HP LSH× 9 (f) HP LSH× 72

(g) CP LSH× 1 (h) CP LSH× 9 (i) CP LSH× 72

(j) SGH × 1 (k) SGH × 9 (l) SGH × 72

(m) DH × 1 (n) DH × 9 (o) DH × 72

Figure 2.8: Histogram of 100 near neighbors found for BFS, hyperplane LSH, cross-polytope LSH, SGH and DH with noisy training dataset of augmented data size 1, 9, and 72; the range of x-axis for all graphs is [0,180] degree.

CHAPTER 3

ROTATION INVARIANT NEURAL NETWORK

3.1 Background Related Work

In the image classification problem, raw inputs could be distorted or transformed on some levels, which affects the classification performance. One of the common solutions to this is learning representative features that are transformation-invariant. Today, there are many well-known techniques designed to capture these crucial features such as the scale-invariant feature transformation (SIFT) [28], the rotation-invariant feature transform (RIFT) [29], and so forth. However, some of these methods are computationally expensive and very sensitive to noise, which makes them less likely to be applied in the cryo-EM problem. Moreover, the learned features may only handle very few variations in the input data and are not very effective in applying for low-powered signals like cryo-EM images. As a result, we need to design an alternative method to extract rotational invariant features in the cryo-EM dataset effectively. The neural network method is well known for its ability of learning expressive features in an more adaptive way based on the task, and it is gradually becoming one of the leading techniques in image classification and recognition problems. Also, with the advent of high-performance computers by using GPU, the structure of the network can go much wider and deeper than ever before, helping to capture more obscure information from the dataset.

To classify cryo-EM images based on the viewing directions, the algorithm should be able to treat images with similar views but different planar orientations as near neighbors. However, the methods we tried in Chapter 2 fail to do that providing with very low accuracy for finding the true near neighbors without image augmentation. Thus, our goal is to build a neural network that is robust to rotated input signals, which is to say, treats the rotated

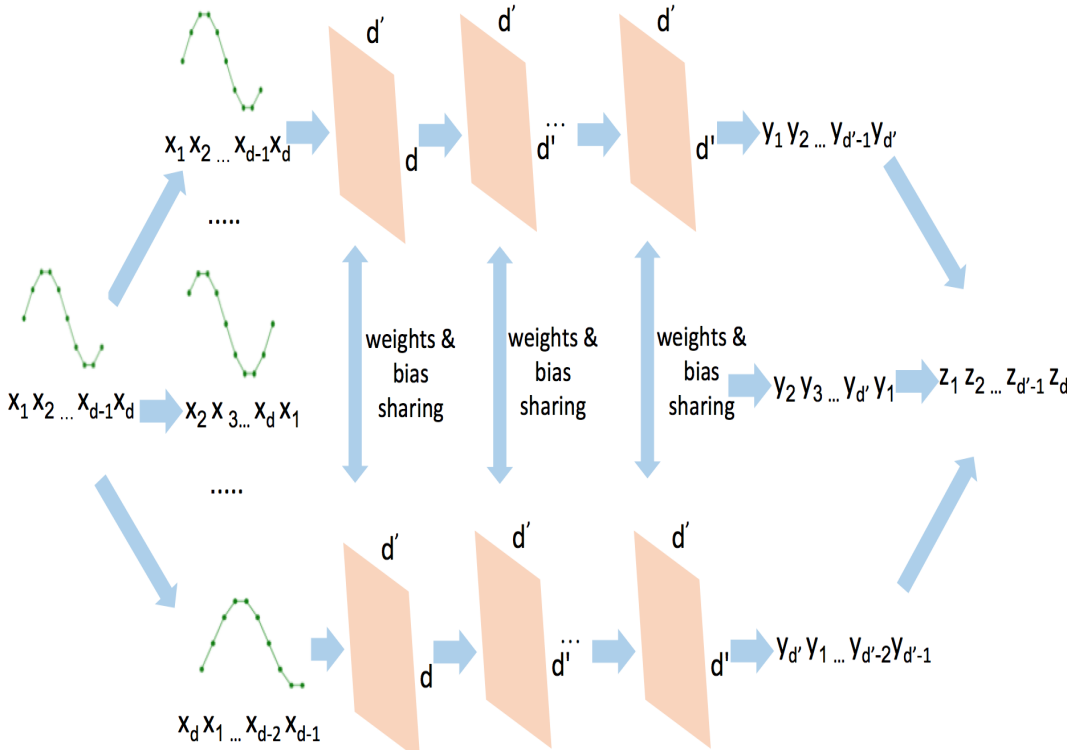


Figure 3.1: The network topology and pipeline of encoding part.

signals as the same signal. In this way, the cryo-EM images with the similar viewing direction but different orientations which looks like rotation copies with each other will share the same features, and later could be classified as near neighbors. This neural network idea is inspired by TI pooling [13], a transformation-invariant pooling for feature learning in convolutional neural networks proposed by Dmitry et al. This method is able to deal with nuisance variances in the image dataset such as rotation and scale differences. It uses parallel siamese architectures [30] for the generated transformation set and then applies the TI pooling operator to obtain the desired invariant features.

3.2 Method Description

The architecture of our network model mainly contains three parts: the encoding network, the average pooling and the decoding network. The network topology and its pipeline of the encoding part and the average pooling is shown in Figure 3.1, and the network structure of the decoding part is

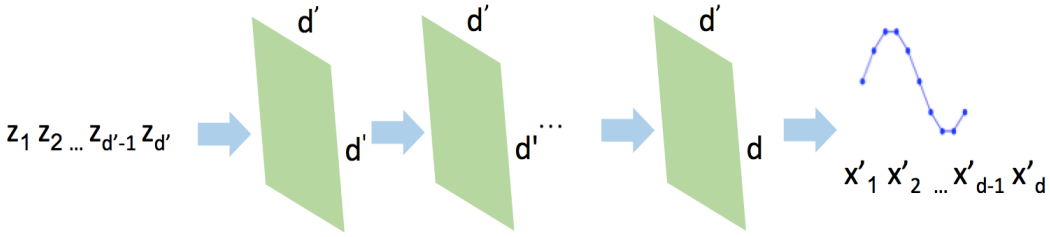


Figure 3.2: The network topology and pipeline of decoding part.

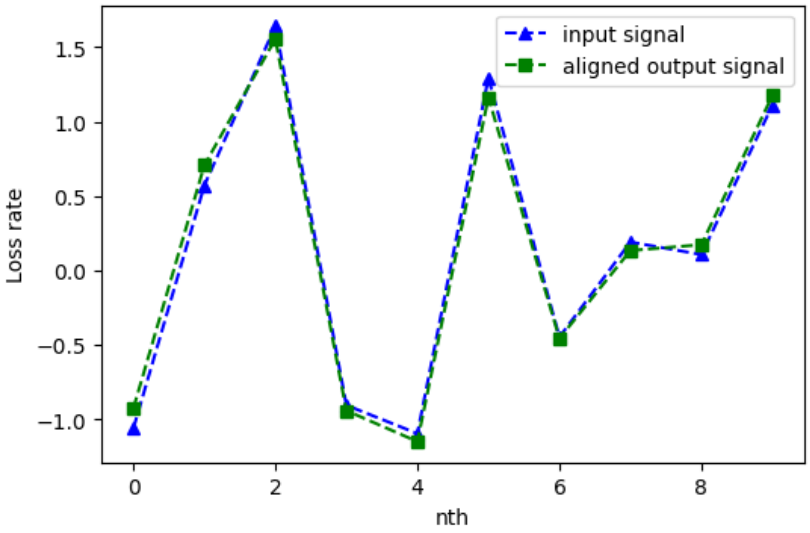


Figure 3.3: The input and aligned reconstructed output signal with length 10. The network contains 3 fully connected layers in the encoding part and 1 fully connected together with 1 linear layer in the decoding part with network size $d \times d \log d$ and $ReLU$ as the activation function.

shown in Figure 3.2. The feature vector attained after the average pooling is rotational invariant, and the decoding part uses this feature vector to reconstruct the input signal. If the reconstructed output signal is trained to be a rotation copy of the input signal, the network model is then proved to have the ability to learn the rotational invariant feature as well as retrieve the original signal.

To start, we feed the simple network model with 1D signal of length d of each entry randomly generated from the normal distribution for simple testing. In the encoding procedure, the input signal is firstly transformed following the rotation transformation that generates d number of shifted/rotated copies for each input signal. The generated rotation set helps the network to "observe" all possible transformations of the input signal, and learn their shared features. These newly generated signal instances are passed through a sequence of encoding layers. The encoding layer is fully connected layer. In each fully connected layer, each input instance x multiplies with a matrix W followed by adding a bias b as $y = Wx + b$. The output y is then passed through a layer that applies an activation function element-wisely. For example, the activation function ReLU will apply $f(x) = max(0, x)$ to each entry of y. The reason we choose the fully connected layer is that this type of layer aims for global feature extraction, and the previous rotation transformation is more of a global transformation than local transformation. Moreover, to capture more information from sets of rotational instances, the size of each layer is $d' \times d'$, where d' is often chosen to be larger than the input dimension d(see Section 3.3.2 for more details). The size of the first layer should be $d \times d'$ to align the dimensionality of the network.

In the next step, the average pooling is applied for every set of rotational instances in order to generate the rotational-invariant features. The average operator forces the network to treat each output of a rotated copy equally and make the feature rotational robustness. The feature vector obtained for each input signal is of length d' as defined in the previous paragraph hope that larger feature length could help to capture more information of the rotated copies fo the input signal, and be used in the later reconstruction step. The feature vectors is then passed to the decoding network. The decoding network is also composed of fully connected layers of size $d' \times d'$, except for the last one which should be $d' \times d$ to transform signal back to the original input length d. Moreover, the ReLU layer is not required to be added at the end

Table 3.1: Batch size vs. Testing loss rate

		Testing Loss Rate						
Batch size		1	5	20	50	100	200	500
Signal length	10	0.148	0.0922	0.0408	0.0749	0.0426	0.0446	0.00143
	41	0.769	0.571	0.356	0.054	0.022	0.0446	0.0461

Note: The network contains 3 fully connected layers in the encoding part and 1 fully connected together with 1 linear layer in the decoding part with network size $d \times d \log d$ and $ReLU$ as the activation function.

of the decoding network.

The loss function for this neural network is the average over the minimum difference between the input and the rotational copies of the corresponding output, which is defined as

$$L = \frac{1}{N} \sum_{n=1}^{N} \min_{\boldsymbol{R}} ||x - \boldsymbol{R}\hat{x}||^2 \tag{3.1}$$

where N is the batch size, x is the input sample, $\hat{x}$ is the corresponding reconstructed output and $\boldsymbol{R}\hat{x}$ represents the rotation copies of the output. In our experiment, d number of rotation copies are generated for the output signal. If the training is efficient, one of the rotated copies of the reconstructed output will be close to the original input signal. Then, it proves that the intermediate features obtained is rotational invariant. An example of the input and reconstructed output for a 1D signal with length 10 is shown in Figure 3.3. The output signal is already aligned with respect to the input signal, and it is easy to tell that they are very similar. The whole procedure is trained by the back propagation, and the optimizer in this model is Adam.

3.3 Experiments and Results

In the experiment, we start with testing 1D input signals instead of directly testing with 2D input signals because the computation is more complicated on 2D image signals and if the practicability of the 1D model is guaranteed, we could then further extend to a 2D model with more confidence. Signals with length 10 and 41 are tested in the following experiments.

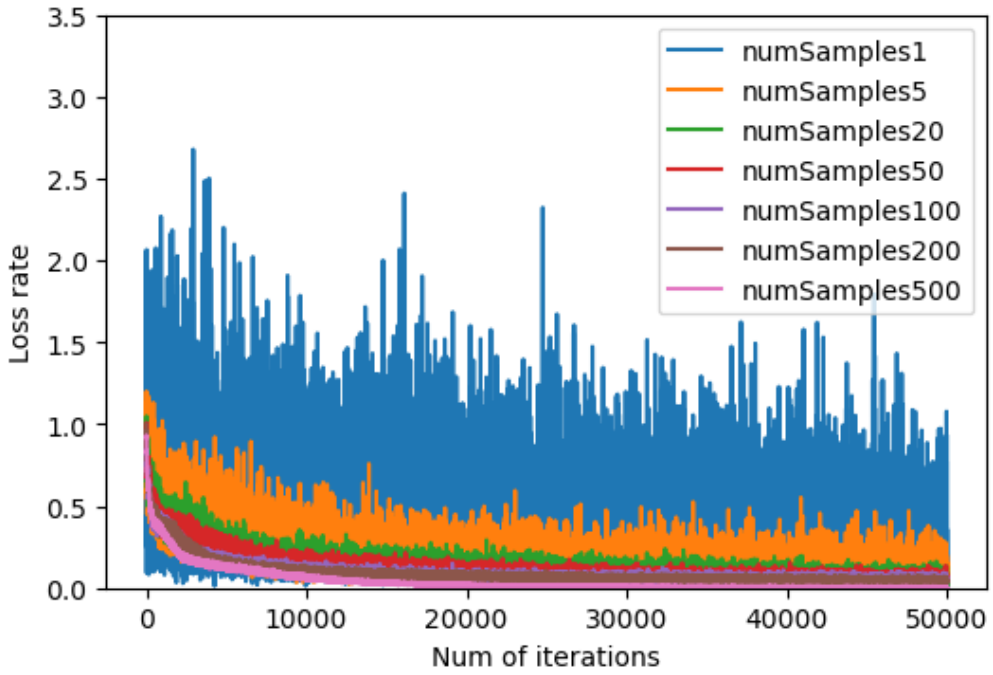


Figure 3.4: Batch size vs. Training loss rate. The network contains 3 fully connected layers in the encoding part and 1 fully connected together with 1 linear layer in the decoding part with network size $d \times d \log d$ and $ReLU$ as the activation function. The input signal length is 10.

3.3.1 Training Batch Size

Training batch size is the number of input samples passed through the neural network for each training iteration. If the batch size is too small, the estimate of the loss function gradient will be less accurate, and the loss value may fluctuate violently along with the number of iterations. On the contrary, if the batch size is too large, too much memory space will be taken when training the model. Therefore, there is a trade-off between the converge trend and the memory space issue. In this section we seek to identify the ideal batch size for the model. Figure 3.4 shows the training loss changing with the number of iterations for signals with length 10, and Table 3.1 shows the test losses for both signals of length 10 and 41.

From the experimental results, we could see that generally speaking, the larger the batch size is, the faster the loss will converge. It is also observed that the loss rate of less batch size fluctuates a lot, while larger batch size gives more consistent and smoother loss rate of which performance looks much better. However, in order to save memory space, too large a batch size should not be used. Since the width of the network layer is larger than the dimension of the input signal, and we are projecting to a higher-dimensional space, large batch size could be very space consuming. Sometimes, it could lead to memory issues in the training procedure. Furthermore, it is noticed that for the input signal with dimension 41, increasing the batch size does not improve the final result. On the contrary, the performances of batch size 200 and 500 are slightly worse than the performance of a batch size of 100. Therefore, the optimal batch size we choose for our simple model is 100.

3.3.2 Network Architecture

The performance of the neural network varies along with its architecture. Primarily, there are two ways to adapt the structure of the neural network: the network width and the network depth. By properly adjusting these two factors, a better performance is achievable. The larger or deeper the neural network is, the more neurons are needed to be learned in the training procedure and the more expensive the computation could be, while a small number of neurons might not be helpful for capturing enough feature information.

Table 3.2: Network width vs. Testing loss rate

		Testing loss rate		
Network width		$2 \times d$	$8 \times d$	$\log d \times d$
Signal length	10	0.078382	0.010689	0.042621
	41	0.173	0.0189	0.0219

Note: The network contains 3 fully connected layers in the encoding part and 1 fully connected together with 1 linear layer in the decoding part with $ReLU$ as the activation function. The batch size is 100.

Table 3.3: Network depth vs. Testing loss rate for signals of length 10

		Testing loss rate		
Number of encoding layer(s)		1	2	3
Number of decoding layer(s)	2	0.097751	0.032379	0.019700
	4	0.076060	0.024541	0.032038
	10	0.129096	0.034145	0.095636

Note: The network is of width $d' = d \times \log d$ and activation function $ReLU$. The batch size is 100.

Network Width Network width defines the number of neurons per layer. Fixing other parameters in the neural network, we try network width $d' = 2 \times d$, $8 \times d$, and $\log d \times d$, where d is the the input signal length. The encoding part and the decoding part of the model share the same network width. Table 3.2 shows that among the selected network width, the larger it is, the smaller the reconstruction loss. This may caused by the fact that more neurons in the layers helps to extract more informative features related to the rotation copies of the signal.

However, too large a network width leads to two non-negligible problems. The first problem is that wider network requires larger matrix multiplications. Larger memory space would be taken if the input signal length is very long, which may lead to memory issues during the training period. The other problem is that more parameters are needed to be trained per layer, while some of them may be redundant capturing noise information. Training those parameters not only waste time but could also cause an inferior performance. Given the influence factors, network width $d' = \log d \times d$ is chosen for our model.

Table 3.4: Network depth vs. Testing loss rate for signals of length 41

		Testing loss rate		
Number of encoding layer(s)		1	2	3
Number of decoding layer(s)	2	0.371893	0.051112	0.021911
	4	0.531867	0.359449	0.453644
	8	0.593307	0.524424	0.711505

Note: The network is of width $d' = d \times \log d$ and activation function $ReLU$. The batch size is 100.

Network Depth Network depth is the number of layers used in the network model. Since our model consists of the encoding part and the decoding part, different combinations of depths for the encoding and decoding network are tested in the experiment. The results in Table 3.3 and Table 3.4 show that more number of encoding layers could help to improve the model performance. This is possibly due to the fact that encoding network depth needs to be large enough to keep the useful feature information from the rotated images. Conversely, we can see that when the number of decoding layers increases for either a signal with length 10 or 41, the performances do not improve much. One possible reason is that the decoding network can be treated as an inverse function of generating the reconstructed input signal from the trained invariant features. However, the reconstructed signal is one of the possible rotation copy of the input signal. Therefore, this inverse function is not unique, which causes the difficulty of training the neural network. Yet, the real reason behind this needs to be further explored.

The combination that gives us a rather good performance consists of an encoding part with three fully connected layers and a decoding part with one fully connected layer and one linear layer. In summary, deeper encoding network could help to capture more crucial features of the rotation copies of input signals, while too many decoding layers are less helpful to improve the performance of the model.

$$ReLU(x) := \begin{cases} x, & \text{if } x \geqslant 0 \\ 0, & \text{if } x < 0 \end{cases} \tag{3.2}$$

$$ReLU^2(x) := \begin{cases} x^2, & \text{if } x \geqslant 0 \\ 0, & \text{if } x < 0 \end{cases} \tag{3.3}$$

Table 3.5: Activation function vs. Testing loss rate for signals of length 10

		Testing loss rate			
Activation functions		ReLU	$ReLU^2$	$ReLU^3$	tanh
Signal length	10	0.045094	0.160807	1.445381	0.067793
	41	0.052421	0.584755	NAN	0.026222

Note: The network is of width $d' = d \times \log d$, and the batch size is 100.

$$ReLU^3(x) := \begin{cases} x^3, & \text{if } x \geqslant 0 \\ 0, & \text{if } x < 0 \end{cases} \tag{3.4}$$

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{3.5}$$

3.3.3 Activation Function

An activation function is applied to the output produced by a neuron. It determines whether the specific neuron will be activated for the next coming layer and the way it will be activated. Activation functions introduce non-linear mappings between the input and output response helping to learn complicated features in the neural network. In the experiment, we first try *ReLU* (3.2) as the activation function for all activation layers, which is considered as the most commonly used activation function. Later, we also test with other three activation functions including $ReLU^2$ (3.3), $ReLU^3$ (3.4), and the hyperbolic function *tanh* (3.5). Table 3.5 shows that the *ReLU* function outperforms the other three functions in general. However, it is noticed that for the signal with length 41, the *tanh* function gives the best performance. Therefore, when dealing with signals of longer length, we may need to further compare the results from *ReLU* with those from *tanh*.

3.3.4 Decoding Methods Comparison

From Section 3.3.2, we find that it is hard to improve the performance of the decoding network, so we think about using other nonlinear-programming method: Quasi-Newton method to do the decoding. This enables us to have

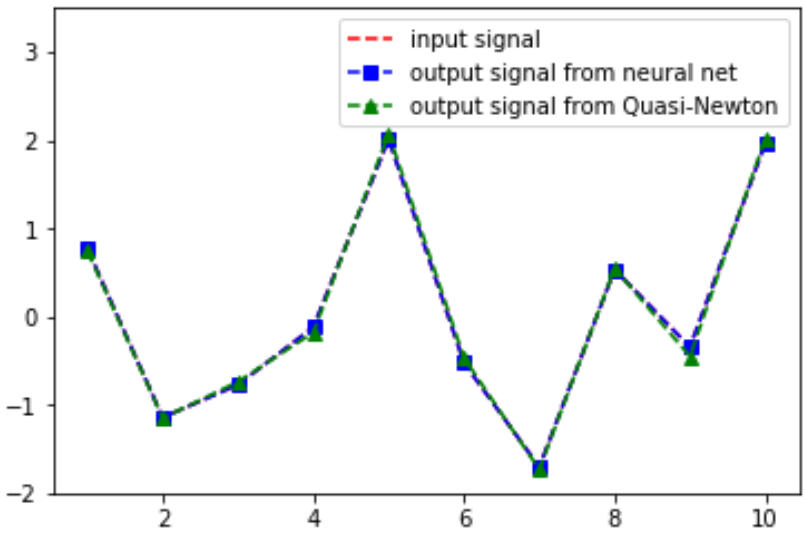


Figure 3.5: Comparison between output results from the decoding network and the Quasi-Newton algorithm for a signal of length 10.

a better understanding about how hard it is to train the decoding part of our model for reconstructing input signals from the rotation invariant features.

The Quasi-Newton method is an algorithm for finding either local maxima or minima of a function $f(x)$. This method is based on Newton's method, but instead of calculating the Hessian matrix of second derivatives of $f(x)$, the Hessian is updated by successive gradient vectors analysis. In each iteration, x is updated by $x_{k+1} = x_k + \alpha\Delta x$, where t is a pre-determined step size, and $\Delta x = -H_k^{-1} \bigtriangledown f(x^{k-1})$ is the Quasi-Newton direction. In a multidimensional problem like our model, Quasi-Newton method is a preferred method to be tested first. In the experiment, we define the feature extraction function of our model $g(x) = \mathbb{E}_{\boldsymbol{R}}(h_{en}(\boldsymbol{R}x, W, b))$, where $\boldsymbol{R}$ is a rotation operator, h_{en} is the encoding network function of our model, and W and b are trained parameters form the encoding network. Then we define the loss function $f(x) = \|(y - g(x))\|^2$, where y is the rotational invariant features from the encoding network for an input x. To find the minima point of the loss function, the Quasi-Newton algorithm is applied returning the reconstructed signal. Figure 3.5 shows an example of the input signal, the reconstructed signal from the Quasi-Newton method model as well as the reconstructed signal from the decoding network. Though the difference is minimal in the graph, the loss rate we got for the Quasi-Newton method is 1.6852e-04 and the loss rate for the decoding network is 0.1663, and it is obviously that the

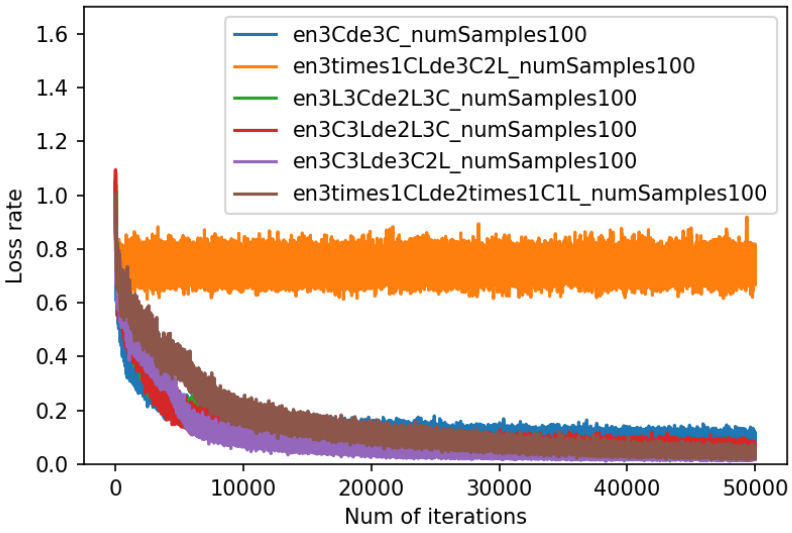


Figure 3.6: Different network combinations. The signal length is 10. The batch size is 100. "en" means the encoding part, and "de" means the decoding part. "xC" means x number of convolutional layers and "xL" means x number of fully connected layers, where x is a constant. The number of filter used in convolutional layers is $\log d$, and the width size of fully connected layers is $d' = d \times \log d$. All layers are using activation function $ReLU$.

Quasi-Newton method model gives a better result.

However, every time a new signal comes in, the Quasi-Newton method has to run thousands of numbers of iterations to get the result, which could be computational expensive. Comparatively, the trained decoding network performed fixed matrix multiplication and addition to the signal to get the output. Thus, we plan to improve our decoding network and approach its performance to that of the Quasi-Newton method in future work.

3.3.5 Add Convolutional Layers

Convolutional layers is most widely used in image recognition to extract features [31]. The convolution operation preserves and learns the spatial relationship between pixels using local regions of the image. In the implementation, a filter or say kernel is sliding over the input image performing convolutions every stride to produce feature maps. The filter refers a matrix

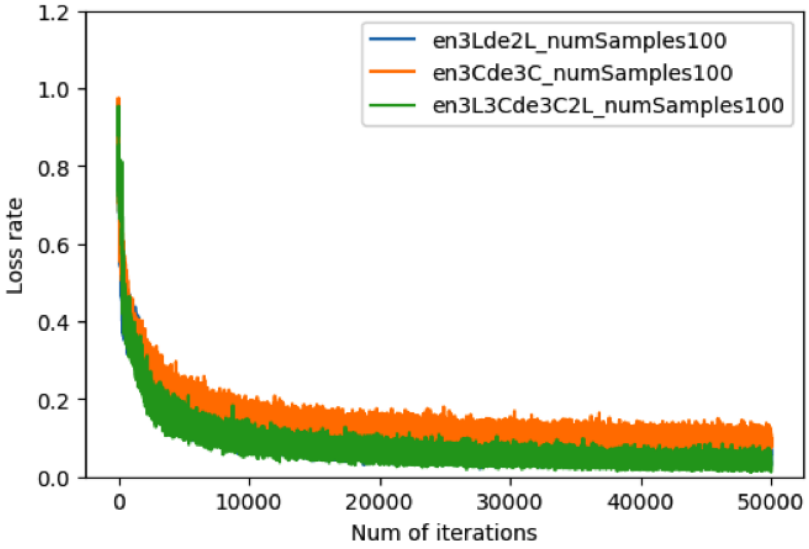


Figure 3.7: Further comparison among three network combinations. The signal length is 10. The batch size is 100. "en" means the encoding part, and "de" means the decoding part. "xC" means x number of convolutional layers and "xL" means x number of fully connected layers, where x is a constant. The number of filter used in convolutional layers is $\log d$, and the width size of fully connected layers is $d' = d \times \log d$. All layers are using activation function $ReLU$.

of which size often is set to 3×3. The term convolution is a mathematical operation performed on two functions to produce a third one. The input image is represented as a form of another matrix. Therefore, when the image convolutes with a filter, a feature map will be produced. Numerous convolutions could be applied on the input image by using different filters to generate different feature maps. The outputs of the convolutions will be passed through the activation function such as $ReLU$ to make the output non-linear. The another term "stride" stands for the step of convolutions conducted each time. If the stride is 1, the filter is sliding one pixel each time and then perform the convolution. It the stride is large, the filter is sliding over the input with a large interval and the overlap between convolutions is small. Since the size of output feature map is always smaller than the input size, padding may be added to prevent the size from shrinking.

In the experiment, we add convolutional layers to our model expecting to extract useful feature information more efficiently in the procedure of training. Based on our current model, the number of filters we use for each convolutional layer is set to be $\log(d)$, where d is the input signal length. The convolutional layers have been assigned to different positions for testing. We try simply using convolutional layers without fully connected layers, adding convolutional layers at the beginning of both encoding part and decoding part, adding convolutional layers at the end of both encoding part and decoding part, adding convolutional layers at the end of encoding part and at the beginning of the decoding part, adding convolutional layers at the beginning of encoding part and at the end of the decoding part as well as adding convolutional layers in the interval of fully connected layers. Results are displayed in Figure 3.6, and adding convolutional layers at the end of encoding part and at the beginning of the decoding part outperforms the others. We then compare them with our model without the implementation of the convolutional layers, and the corresponding result is shown in Figure 3.7. The difference between them is quite small, which means that adding convolutional layers could not effectively improve our model performance. It could be lead by the fact that convolutional layers focus more on local area learning, while our rotation copies are more related to global transformation and convolutional layers could do little to help that. As a result, we decide to use our original model with only fully connected layers and perform further tests on noisy and partially observable signals.

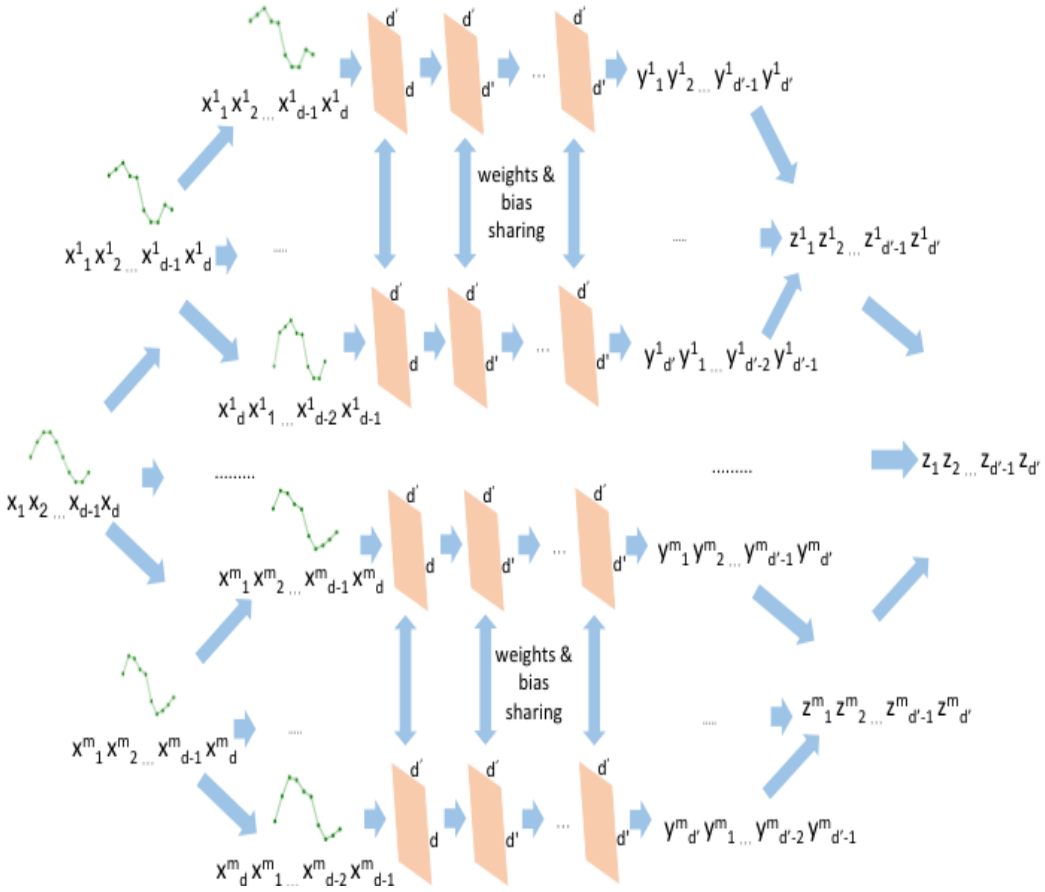


Figure 3.8: The network topology and pipeline of the encoding network with m noisy observations for each signal.

Table 3.6: SNR vs. Testing loss rate with signals of length 10

Batch size	Clean	SNR=100	SNR=10	SNR=1
1	0.075054	0.130631	0.172946	0.384211
5	0.021663	0.087652	0.172946	0.289209
20	0.018975	0.105584	0.107960	0.222037
50	0.010098	0.056381	0.099914	0.204227
100	0.019456	0.045291	0.099984	0.218310

Note: The network contains 3 fully connected layers in the encoding part and 1 fully connected together with 1 linear layer in the decoding part with network width $d' = d \times \log d$ and $ReLU$ as the activation function. The batch size is 100 for all testing.

3.3.6 Noisy Signal

In the section, we test how robust our network model reacts to different levels of SNR. We finally want to apply the neural network model in the SPR problem by searching in the rotational invariant features of the cryo-EM projection images. However, those images are heavily contaminated with noise. Though we tried the preprocessing techniques, such as FBsPCA to de-noise the signal, the performance is not satisfying. Therefore, we want to observe how well the neural network model could deal with the noisy input signals.

In the experiment, m noisy signals are created for each input signal as the "observable" signals available to the researcher. The noisy sample is generated by adding the input signal with additive white Gaussian noise. Later, each noisy sample is then passed through the network by first generating the rotation transformation instances, and the following procedure is the same as described for the clean dataset model. At the end, we will see how well the network could reconstruct the clean signal based on those noisy image copies. The new pipeline of the encoding network is shown in Figure 3.8.

SNR of 1, 10 and 100 are tried in the experiment as shown in Table 3.6. Results demonstrate that the smaller SNR value is, the larger the loss rate. This makes sense since if SNR is small, the energy of the signal is small compared with the energy of the noise. If the signal is heavily contagious by the noise, a lot of intrinsic information will be discarded and it will be very hard for the network model to reconstruct the original signal back.

Table 3.7: Partial observation length vs. Testing loss rate for original signals of length 10

		Testing loss rate			
Segment length		4	6	8	10
Batch size	20	0.160	0.0484	0.0559	0.0119
	50	0.161	0.0516	0.0138	0.0113
	100	0.1387	0.0353	0.0239	0.0109
	200	0.120	0.0362	0.0163	0.00567
	1000	0.133	0.0313	0.0198	0.00338

Note: The network contains 3 fully connected layers in the encoding part and 1 fully connected together with 1 linear layer in the decoding part with network width $d' = d \times \log d$ and $ReLU$ as the activation function.

Table 3.8: Partial observation length vs. Testing loss rate for original signals of length 41

		Testing loss rate				
Segment length		6	10	16	21	31
Batch size	50	0.675	0.532	0.371	0.295	0.162
	100	0.622	0.540	0.350	0.299	0.149
	200	0.653	0.523	0.375	0.177	0.111
	500	0.662	0.522	0.349	0.205	0.124

Note: The network contains 3 fully connected layers in the encoding part and 1 fully connected together with 1 linear layer in the decoding part with network width $d' = d \times \log d$ and $ReLU$ as the activation function.

Furthermore, although SNR is very large such as 100 in the experiment, the test loss is still noticeable and four to five times larger than the test loss of the clean data. Thus, further improvements are required to be done for our network model for reconstructing from noisy signals.

3.3.7 Segmented Signal

According to the Fourier projection-slice theorem shown at the top of Figure 3.9, it is possible to reconstruct 3D model of a molecule of which cryo-EM projection images are randomly oriented. A representative reconstruction method is called Kam's method [32] using methods of moments and non-

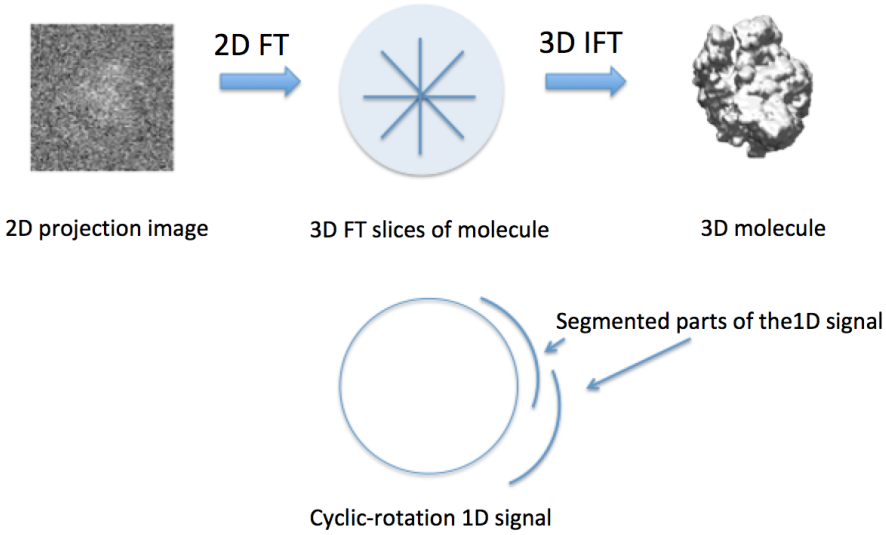


Figure 3.9: The top pipeline shows the reconstruction 3D model from 2D images by using the Fourier projection-slice theorem, and the bottom shows how 1D signal is related to this pipeline.

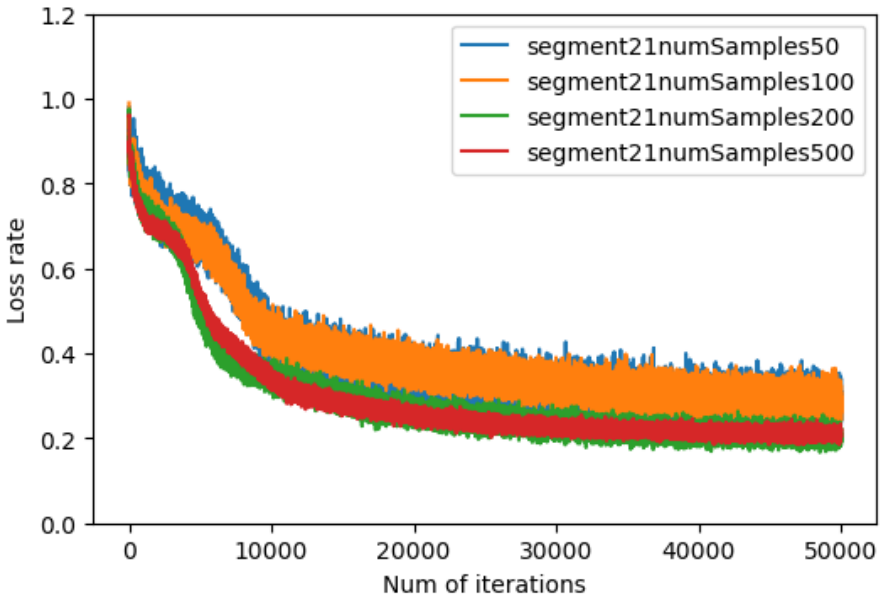


Figure 3.10: Partially observable signal of original length 41 with segment length 21 for different batch sizes. The network contains 3 fully connected layers in the encoding part and 1 fully connected and 1 linear layer in the decoding part with network width $d' = d \times \log d$ and activation function $ReLU$.

linear programming. We start with testing 1D signal, since the segmented cyclic-rotation parts of the 1D signal can be treated the way as the 2D projection images for the whole 3D structure of the molecule demonstrated at the bottom of Figure 3.9. So in this section, we test whether the network could fully reconstruct the input signal if the signal is only partially observable so as to test the possibility of reconstructing the molecule model based on a large set of projection images.

In the experiment, the length of the instances d in the rotation transformation set is truncated to the segmentation length s. Those newly generated rotational instances are treated as the partially observable signals, and the model needs to reconstruct the original full length signal based on those observable instances. We test with 1D signal with length 10 and 41 as shown in Table 3.7 and Table 3.8. From tables, in general, we can say that larger batch size helps to improve the test loss rate, while the improvement will not be very obvious if the batch size is above 200. Moreover, from Figure 3.10 we see that after 20000 iterations, the trend for decreasing the loss rate becomes much smaller and the loss stays around 0.2. It may be the reason that the loss function got stuck in a local minimum. The loss itself is quite large considering the magnitude of the original data which is generated from i.i.d normal distribution. Furthermore, we test a different number of decoding layers trying to lower the loss rate. However, it is discovered that increasing the number of decoding layers does not help improve the model performance. And this problem becomes more challenging along with the increment of the length of the input signal. Therefore, we plan to explore other forms of decoding architecture in the future.

3.4 Summary

It is shown that the longer the signal length, the harder it will be to train the network model. It could be due to the fact that feature information is not informative enough or the decoding network is not effective in reconstruction. At the same time, increasing the depth of the decoding network does not help with the training procedure. The reason behind this needs to be identified and further improvements are needed for the decoding part of our model.

From the above experiments, we find three layers of encoding network and

two layers of decoding network with network layer size $(d \times \log d) \times (d \times \log d)$ together with activation function $ReLU$ could give us a comparatively better result for the clean signals with length 10 and 41. The batch size we prefer to use in this model is 100, which is large enough for our training procedure. What is more, we found that the updated network model is not very robust in response to the noise. It is observed that if SNR is small, the signal will be very hard to reconstruct, since most of the key information is destroyed. As for the segmented signals, if the original length is long, the reconstruction loss could be unsatisfying. Even though the original length is small, if the segmentation length is small, for example, half of the original data length, the reconstruction loss may still be large. More studies that concentrate on these issues are recommended.

CHAPTER 4

CONCLUSION

4.1 Discussion

In Chapter 2, we apply four different k-nearest neighbor search algorithms including hyperplane locality sensitive hashing, cross-polytope locality sensitive hashing, unsupervised stochastic generative hashing, and unsupervised deep hashing separately on cyro-EM projection image dataset finding near neighbors for each query image data based on the viewing direction. When analyzing the performances of these algorithms, we not only compare their accuracies for near neighbors, but also the construction time/training time as well as the data query time. For the clean dataset, the accuracy defined is based on the ground truth table obtained from the quaternion matrix. According to the results, cross-polytope LSH is the fastest in the construction and query time among all the methods. Furthermore, the accuracy of cross-polytope LSH is only slightly inferior to that of hyperplane LSH, which is still comparatively high. Therefore, we claim that cross-polytope LSH could be a choice for classifying near neighbors for the clean cryo-EM dataset.

As for the noisy image dataset, the accuracy is redefined such that if the angle difference between the query and the candidate data found is less than a certain degree, we could take them as near neighbors. In this part of the experiment, we try three preprocessing strategies: the traditional PCA, the FBsPCA and the bispectrum method. In general, the bispectrum method outperforms other two methods considering the accuracy for each searching algorithm, and the fact that it does not need to preprocess the dataset with the image augmentation method, which could be very time and space consuming.

In Chapter 3, we demonstrate a new neural network model which trains rotation-invariant features. This model has an end-to-end architecture con-

sisted of encoding and decoding networks and the average pooling in the middle procedure. Features are obtained after the average pooling, and the reconstructed signal is produced at the end of the decoding network. The reconstructed signal is in a form of the rotation copy of the input sample. After testing the network model with signals of different lengths, we find that the longer the signal, the harder it will be to reconstruct it back. The model with three layers of encoding network and two layers of decoding network gives the lowest testing loss rate so far. It is shown that more numbers of decoding layers may not improve the reconstruction performance further, and two decoding layers is the best architecture so far. Moreover, increasing the network width could help to decrease the loss rate. However, the size of network could not be set too large since it requires more memory space which may cause the memory problems during the training procedure. In addition, for the noisy dataset, we find that enlarging the number of training iterations could converge the loss rate to a very low value. However, as for the segmented dataset, though the segmentation length is larger than half of the original signal length, it is still hard to train and reconstruct the input signal back. For this problem, other strategies may need to be applied to improve the network capability.

4.2 Comparisons with Previous Work

The four searching algorithms we try are the most recent methods that are used in near neighbor search problems. Hyperplane LSH is very efficient in constructing hash tables and data query, while cross-polytope LSH is even faster but sometimes gives slightly lower accuracy for finding near neighbors. Initially, we hoped the neural network methods - unsupervised SGH and DH could capture the rotation-invariant feature information of the dataset, but they could not. Since it could be very computation intensive given the large cryo-EM image dataset obtained at hand, LSH methods will be a preference. Applied with the LSH method, even the dataset augmented by 72 times could be mapped into hash tables within several seconds, and the searching time for each query data is approximately microseconds.

There are many algorithms available to capture transformation invariant features. However, some of them are quite computational expensive such

as SIFT, while others could only be applied to certain forms of dataset. Since the neural network method is considered to be able to learn features in a more expressive way, while methods we try like unsupervised SGH and unsupervised DH are not able to do capture the features we desired, we try to develop a neural network that could handle the rotational invariant feature extraction. Our model not only could extract the rotational invariant features, but also reconstruct the signal from the generated features in a form of a rotation copy of the original input signal, which proves the uniqueness and robustness of the rotation invariant features. It is different from the TI pooling method which only creates the transformation invariant features without reconstructing the signal back. On the other hand, we also test with noisy and segmented dataset trying to demonstrate the flexibility of the model in reaction to these imperfect dataset. It shows that the performance is not that satisfying for the segmented signal. Its loss rate converges very slowly and finally stays at a high loss value.

4.3 Future Work

For Chapter 2, we can see that for the clean dataset, all the methods work well under the case that the augmentation size is large. However, when applying algorithms to the noisy image dataset, none of them could give satisfying performance even with a large augmentation size. Though the de-noising bispectrum method gives the best result compared to the traditional PCA and FBsPCA, the accuracy for each algorithm is around 70%, which is rather low. The SNR of the noisy dataset is only 0.1, and this may be the best SNR in the real cryo-EM dataset we can achieve. In the future, we need to find a more efficient strategy to deal with datasets with small SNR, and then we could use the de-noised dataset in the nearest neighbor search algorithm.

For the new network model, many modifications are recommended to improve its performance. Since the length of input data could be very large, the current toy model is too simple to handle the long signals. Moreover, in the cryo-EM problem, data samples are quite noisy. The model should be very robust to the noise after observed copies of noisy data are given. However, the model only works well under the situation when the signal length is

small. Furthermore, since we are considering to use this model to directly reconstruct a 3D model from its 2D projection images, we treat the segmented signals as images and the output full length signal as newly generated information about the 3D model. However, it is shown that the model is unable to reconstruct original signal from the segmented data signals effectively.

In the future, we recommend adjustments to our model to cope with longer signals and noisy signals. Then we can move to dealing with 2D image signals generating rotational invariant features for the LSH method which could quickly search for images of similar views. In addition, further improvements to the model are suggested for solving the segmented signal issues, which could be quite challenging.

REFERENCES

[1] J. Frank, *Three-Dimensional Electron Microscopy of Macromolecular Assemblies: Visualization of Biological Molecules in Their Native State*. Oxford University Press, 2006.

[2] Z. Zhao and A. Singer, "Rotationally invariant image representation for viewing direction classification in cryo-EM," *Journal of Structural Biology*, vol. 186, no. 1, pp. 153–166, 2014.

[3] R. Weber, H.-J. Schek, and S. Blott, "A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces," in *VLDB*, vol. 98, 1998, pp. 194–205.

[4] A. Andoni and P. Indyk, "Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions," in *Foundations of Computer Science, 2006. FOCS'06. 47th Annual IEEE Symposium on*. IEEE, 2006, pp. 459–468.

[5] J. Leskovec, A. Rajaraman, and J. D. Ullman, *Mining of Massive Datasets*. Cambridge University Press, 2014.

[6] M. S. Charikar, "Similarity estimation techniques from rounding algorithms," in *Proceedings of the Thiry-Fourth Annual ACM Symposium on Theory of Computing*. ACM, 2002, pp. 380–388.

[7] A. Andoni, P. Indyk, T. Laarhoven, I. Razenshteyn, and L. Schmidt, "Practical and optimal LSH for angular distance," in *Advances in Neural Information Processing Systems*, 2015, pp. 1225–1233.

[8] A. Andoni, P. Indyk, H. L. Nguyen, and I. Razenshteyn, "Beyond locality-sensitive hashing," in *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*. Society for Industrial and Applied Mathematics, 2014, pp. 1018–1028.

[9] A. Andoni and I. Razenshteyn, "Optimal data-dependent hashing for approximate near neighbors," in *Proceedings of the Forty-Seventh Annual ACM Symposium on Theory of Computing*. ACM, 2015, pp. 793–801.

[10] L. Qin, W. Josephson, W. Zhe et al., "Efficient indexing for high-dimensional similarity search," in *Proceeding-VLDB 07 Proceedings of the 33rd International Conference on Very Large Data Bases*, 2007.

[11] B. Dai, R. Guo, S. Kumar, N. He, and L. Song, "Stochastic generative hashing," *arXiv preprint arXiv:1701.02815*, 2017.

[12] V. Erin Liong, J. Lu, G. Wang, P. Moulin, and J. Zhou, "Deep hashing for compact binary codes learning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 2475–2483.

[13] D. Laptev, N. Savinov, J. M. Buhmann, and M. Pollefeys, "Ti-pooling: Transformation-invariant pooling for feature learning in convolutional neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 289–297.

[14] M. Raginsky and S. Lazebnik, "Locality-sensitive binary codes from shift-invariant kernels," in *Advances in Neural Information Processing Systems*, 2009, pp. 1509–1517.

[15] Y. Gong, S. Lazebnik, A. Gordo, and F. Perronnin, "Iterative quantization: A procrustean approach to learning binary codes for large-scale image retrieval," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 12, pp. 2916–2929, 2013.

[16] M. Van Heel and J. Frank, "Use of multivariates statistics in analysing the images of biological macromolecules," *Ultramicroscopy*, vol. 6, no. 1, pp. 187–194, 1981.

[17] M. Lindenbaum, M. Fischer, and A. Bruckstein, "On Gabor's contribution to image enhancement," *Pattern Recognition*, vol. 27, no. 1, pp. 1–8, 1994.

[18] G. H. Golub and C. Reinsch, "Singular value decomposition and least squares solutions," *Numerische Mathematik*, vol. 14, no. 5, pp. 403–420, 1970.

[19] R. Hilai and J. Rubinstein, "Recognition of rotated images by invariant Karhunen–Loéve expansion," *JOSA A*, vol. 11, no. 5, pp. 1610–1618, 1994.

[20] M. Uenohara and T. Kanade, "Optimal approximation of uniformly rotated images: Relationship between Karhunen-Loeve expansion and discrete cosine transform," *IEEE Transactions on Image Processing*, vol. 7, no. 1, pp. 116–119, 1998.

[21] Z. Zhao, Y. Shkolnisky, and A. Singer, "Fast steerable principal component analysis," *IEEE Transactions on Computational Imaging*, vol. 2, no. 1, pp. 1–12, 2016.

[22] B. M. Sadler and G. B. Giannakis, "Shift-and rotation-invariant object reconstruction using the bispectrum," *JOSA A*, vol. 9, no. 1, pp. 57–69, 1992.

[23] S. Distributions, "Pseudorandom generators, embeddings and data stream computation," *Piotr Indyk: FOCS*, pp. 189–197, 2000.

[24] L. Engebretsen, P. Indyk, and R. O'Donnell, "Derandomized dimensionality reduction with applications," in *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms*. Society for Industrial and Applied Mathematics, 2002, pp. 705–712.

[25] N. Ailon and B. Chazelle, "The fast Johnson-Lindenstrauss transform and approximate nearest neighbors," *SIAM J. Comput.*, vol. 39, no. 1, pp. 302–322, May 2009. [Online]. Available: http://dx.doi.org/10.1137/060673096

[26] Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li, "Multi-probe lsh: Efficient indexing for high-dimensional similarity search," in *Proceedings of the 33rd International Conference on Very Large Data Bases*, ser. VLDB '07. VLDB Endowment, 2007. [Online]. Available: http://dl.acm.org/citation.cfm?id=1325851.1325958 pp. 950–961.

[27] M. Á. Carreira-Perpiñán and R. Raziperchikolaei, "Hashing with binary autoencoders," in *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*, 2015. [Online]. Available: https://doi.org/10.1109/CVPR.2015.7298654 pp. 557–566.

[28] D. G. Lowe, "Object recognition from local scale-invariant features," in *Proceedings of the International Conference on Computer Vision-Volume 2*, ser. ICCV '99. Washington, DC, USA: IEEE Computer Society, 1999. [Online]. Available: http://dl.acm.org/citation.cfm?id=850924.851523 pp. 1150–.

[29] S. Lazebnik, C. Schmid, and J. Ponce, "Semi-local affine parts for object recognition," in *BMVC*. BMVA Press, 2004, pp. 1–10.

[30] J. Bromley, I. Guyon, Y. LeCun, E. Säckinger, and R. Shah, "Signature verification using a "siamese" time delay neural network," in *Advances in Neural Information Processing Systems 6*, J. D. Cowan, G. Tesauro, and J. Alspector, Eds. Morgan-Kaufmann, 1994, pp. 737–744. [Online]. Available: http://papers.nips.cc/paper/769-signature-verification-using-a-siamese-time-delay-neural-network.pdf

[31] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1097–1105. [Online]. Available: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

[32] A. Singer, "Mathematics for cryo-electron microscopy," *arXiv preprint arXiv:1803.06714*, 2018.